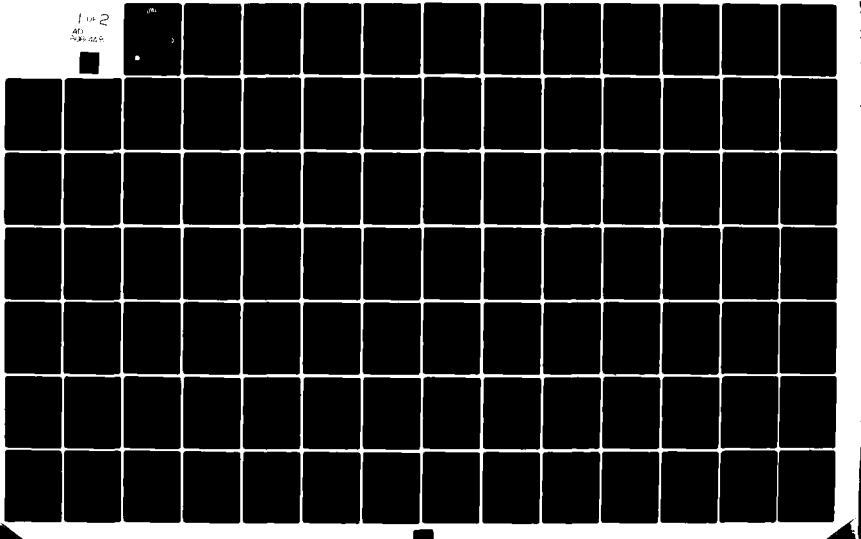
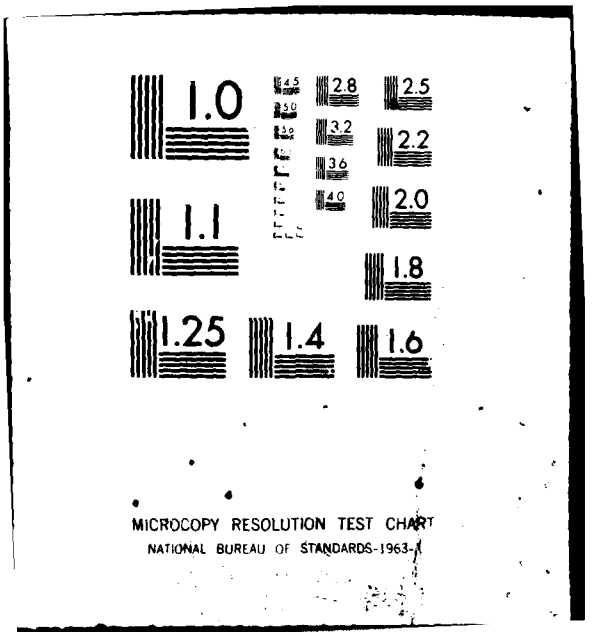
AD-A081 448 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 12/1
GEOMETRIC TRANSFORMS FOR FAST GEOMETRIC ALGORITHMS.(U)
DEC 79 K Q BROWN N00014-76-C-0370
UNCLASSIFIED CMU-CS-80-101 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Geometric Transforms for Fast Geometric Algorithms

Kevin Q. Brown

24 December 1979

See 1473 in back.

DEPARTMENT
of
COMPUTER SCIENCE

Carnegie-Mellon University

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

CMU-CS-80-101

Geometric Transforms for Fast Geometric Algorithms

Kevin Q. Brown

24 December 1979

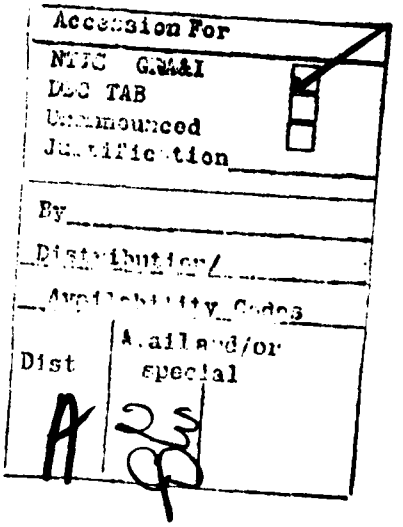
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

Submitted to Carnegie-Mellon University in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was partially supported by the Office of Naval Research under contract number N00014-76-C-0370.

ABSTRACT

Many computational problems are inherently geometrical in nature. For example, cluster analysis involves construction of convex hulls of sets of points, LSI artwork analysis requires a test for intersection of sets of line segments, computer graphics involves hidden line elimination, and even linear programming can be expressed in terms of intersection of half-spaces. As larger geometric problems are solved on the computer, the need grows for faster algorithms to solve them. The topic of this thesis is the use of geometric transforms as algorithmic tools for constructing fast geometric algorithms. We describe several geometric problems whose solutions illustrate the use of geometric transforms. These include fast algorithms for intersecting half-spaces, constructing Voronoi diagrams, and computing the Euclidean diameter of a set of points. For each of the major transforms we include a set of heuristics to enable the reader to use geometric transforms to solve his own problems.



Acknowledgments

The CMU Computer Science Department has provided a rare combination of a very pleasant atmosphere and excellent facilities that I do not expect to find very often for many years. Michael Shamos introduced me to computational geometry. I would especially like to thank Jon Bentley for his encouragement and help on my thesis and other work during the past few years. He has gone beyond the call of duty so many times that I really do not know how to adequately thank him.

Table of Contents

1. Introduction

Many computational problems are inherently geometrical in nature. For example, cluster analysis involves construction of convex hulls of points, LSI artwork analysis requires a test for intersection of sets of line segments, computer graphics involves hidden line elimination, and even linear programming can be expressed in terms of intersection of half-spaces. As larger geometric problems are solved on the computer, grows for faster algorithms to solve them. To obtain fast geometric algorithms a set of tools and techniques has been developed that takes advantage of the structure provided by the geometry. This discipline is known as Computational Geometry. In the following sections we first survey the previous work in computational geometry and then outline the contributions of this thesis.

1.1. History of Computational Geometry

Geometry has been studied for thousands of years but only recently has it been recast in computational form. Shamos [91] describes the history of geometry from the perspective of a computer scientist. Here we will consider only the history of *computational* geometry. There are several problem areas to which much research has been devoted -- construction of convex hulls, intersection problems, closest-point problems, and geometric searching problems. In this section we summarize the major results in each of these areas and also a number of topics that do not fit into these categories.

1.1.1. Convex Hulls

The convex hull of a set of points is a fundamental geometrical structure that arises in a multitude of different problems in the literature and this thesis. Mathematics texts define the convex hull of a set of points S as the smallest convex set that contains all of the points of S. This definition is fine for proving theorems but it does not help us design a fast algorithm.

In 1970 Chand and Kapur [25] produced a convex hull algorithm for N points in K-space. They applied a procedure called "giftwrapping" to obtain a good (but not

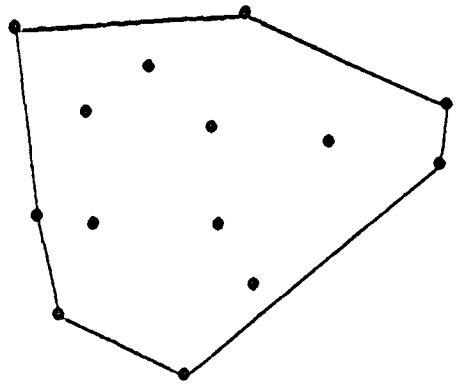


Figure 1-1: Convex hull of a planar set of points.

optimal) algorithm. Graham [48] presented an O(N log N) time planar algorithm in 1972. This is optimal, in the worst-case sense, because an algorithm that constructs a convex hull can be used to sort [91]. (If we require only the vertices of the convex hull, then we can no longer use a convex hull algorithm to sort. Nevertheless, Yao [103] has proven that Ω(N log N) time is still required in the worst case when only quadratic functions of the input are allowed.) Jarvis [54] subsequently applied giftwrapping to the planar problem to obtain an O(VN) time algorithm where V is the number of vertices on the hull. If V is less than O(log N) then Jarvis' algorithm is faster than Graham's. (Preparata [81] later refined Graham's result by constructing an O(N log N) time *real-time* planar convex hull algorithm. Rather than operating on all N points collectively, this algorithm updates the hull in O(log N) time after each point is read.) Preparata and Hong [83] then solved the three-dimensional convex hull problem in O(N log N) time. In four dimensions, however, there is an $\Omega(N^2)$ lower bound because the convex hull can have $\theta(N^2)$ edges ([49], p.193).

The lower bounds above apply only to the worst-case. If the expected number of points on the convex hull is sublinear, then faster expected-time algorithms are possible. Floyd [40] and Eddy [39] independently discovered a planar convex hull algorithm with O(N) expected-time when the N points are drawn from a uniform

distribution over a convex region. Bentley and Shamos [16] improved that result to include any distribution for which the expected number of points on the hull is $O(N^p)$ for some $p < 1$. Furthermore, their result extends to O(N) expected-time in three-space while maintaining an O(N log N) worst-case time. For K dimensions we may still construct the convex hull in O(N) expected-time if the K coordinates are drawn from independent distributions. In this case the expected values for the number of maxima and the square of the number of maxima are only $O(\log^{K-1} N)$ [10] and $O(\log^{2(K-1)} N)$ [30], respectively. Even though the worst-case number of edges, faces, etc. of a K-dimensional convex hull of N vertices grow exponentially with K, the expected size of the convex hull is still only a power of log N.

1.1.2. Intersection Problems

Intersection problems have also received a great deal of attention in recent years. They occur in a variety of areas including computer graphics, architectural data bases, printed circuit design, and even linear programming [94, 12, 2, 65]. Shamos and Hoey [90, 91] constructed several fundamental intersection algorithms including a linear time intersection of two convex polygons, which is applied recursively in their O(N log N) time algorithm for intersecting N half-planes. Hoey's O(N log N) time algorithm [94] to determine if any two of N line segments intersect has been extended by Bentley and Ottmann [12] to report all K intersecting pairs in O(N log N + K log N) time. (Brown [24] has reduced the storage requirement of Bentley and Ottmann's algorithm from O(N + K) to O(N).) If all segments are either horizontal or vertical, then the number of intersections K can be counted in O(N log N) time and reported in O(N log N + K) time. (See [4, 19, 98] for rectangle intersection problems and algorithms.) Finally, Zolnowsky [106] and Preparata and Muller [84] have applied geometric transforms to produce three algorithms for intersecting N three-dimensional half-spaces in O(N log N) time.

1.1.3. Closest Point Problems

Closest point problems arise in cluster analysis, pattern recognition, and, in particular, construction of minimum spanning trees [93, 3]. Dobkin and Lipton [34, 35] used an interesting duality transform to prove an $\Omega(N \log N)$ time lower bound for the element-uniqueness problem[1] under a model of computation that allows (<,=,>) comparisons of linear functions of the input.[2] This proves an $\Omega(N \log N)$ time lower bound for the problem of finding the two closest of N points. Their model of computation, however, allows only comparisons between linear functions of the input. With a stronger model of computation algorithms faster than O(N log N) time are possible. Fortune and Hopcroft [41] showed that if the floor function is allowed, the two closest points can be found in O(N log log N) time in the worst case. Previously, Rabin [86] and Yuval [104] had given O(N) *expected-time* algorihms for the K-dimensional closest pair problem.

Meanwhile, Bentley (and others) worked on multi-dimensional nearest neighbor problems [9, 11, 14, 17] and he invented a data structure called a K-D tree to solve them efficiently. Bentley's thesis [3] employed a strategy called *multi-dimensional divide-and-conquer* with which he obtained the first sub-quadratic algorithms for several multi-dimensional closest point problems. His thesis is also a good source for learning about algorithm design -- rather than simply presenting the finished product he displays the algorithm design process and at the conclusion presents a list of heuristics to use in designing algorithms.

Shamos and Hoey [89, 91, 93] created an O(N log N) time divide-and-conquer algorithm for constructing a *Voronoi diagram* of N planar points. A Voronoi diagram

---

[1] The element uniqueness problem is to determine whether all N elements of an unordered multiset are unique.

[2] If the allowed (<,=,>) comparisons are restricted to be between the N elements themselves -- no linear functions of the input -- then an $\Omega(N \log N)$ time lower bound is easy to prove. This is because no ordering of the N elements other than a total ordering can guarantee that no two elements are equal. Construction of a total ordering, however, requires a sort, which costs $\Omega(N \log N)$ time.

(to be described in detail in Section 4.1) contains all of the necessary proximity information to solve efficiently a number of closest-point problems including construction of a Euclidean minimal spanning tree, a proper straight-line triangulation of the N points, and the nearest neighbor problem. Bentley, Weide, and Yao [18] have extended the techniques of Weide's thesis [99] to obtain a linear expected-time algorithm for constructing a planar Voronoi diagram. The only conditions are that the probability density of the underlying distribution be bounded both above by a constant and (below) away from zero over some finite region.

Bentley and Friedman [8] describe a heuristic solution for a minimum spanning tree algorithm in multi-dimensional Euclidean space and Yao [102] has constructed provably subquadratic worst-case MST (and related) algorithms for the L_1, L_2, and L_∞ metrics. (For $K \geq 3$ dimensions his algorithms take $O(N^{2-\alpha(K)}(\log N)^{1-\alpha(K)})$ time where $\alpha(K) = 2^{-(K+1)}$. For the special case of the Euclidean metric in three dimensions this is improved to $O((N \log N)^{1.8})$ time.) In general, though, construction of a minimum spanning tree is a graph problem. Kruskal [63] and Prim [85] give the classical $O(EV)$ time algorithms (improved to $O(V^2)$ time [32]). Both Yao [101] and Cheriton and Tarjan [26] present $O(E \log \log V)$ time algorithms for general graphs of E edges and V vertices, and Cheriton and Tarjan [26] also present an $O(V)$ time algorithm for planar graphs.

1.1.4. Geometric Searching Problems

As we saw above, many closest-point problems have associated searching problems; in this section we summarize a separate class of searching problems that are not related to any particular closest-point problem.

Shamos [89] gives an algorithm that locates a point in a straight-line planar graph of N vertices in $O(\log N)$ time and Dobkin and Lipton [33] and Dewdney [31] extended this technique to K dimensions. Unfortunately, the storage and preprocessing time required by these algorithms are prohibitive -- $O(N^2)$ in the planar case and $O(N^{2^K})$ in the K-dimensional case [102]. Lee and Preparata [66] improved the storage and preprocessing time at the expense of increased

searching time, giving an algorithm with $O(\log^2 N)$ query time, O(N log N) preprocessing time, and O(N) storage. (Shamos and Hoey [89, 93] had achieved these bounds for searching Voronoi diagrams.) Another alternative is Preparata's algorithm with only O(log N) query time but O(N log N) preprocessing time *and storage* [80]. In 1977 Lipton and Tarjan achieved an O(log N) query time with only O(N log N) preprocessing time and O(N) storage [69, 70]. The point location problem is generalized to the location of a *set* of points in [68, 82].

Kung, Luccio, and Preparata [64] worked on the problem of finding the maxima of a set of N vectors in K-space. (A vector is maximal if none of the other N - 1 vectors are greater in all K coordinates.) Using divide-and-conquer, they constructed an algorithm that finds the maxima in O(N log N) time in two dimensions and $O(N (\log N)^{K-2})$ time in $K \geq 3$ dimensions. They also proved an $\Omega(N \log N)$ lower bound for the problem. Maxima are important because of their relationship to convex hulls and ECDF's (to be described). Bentley, Kung, Schkolnick, and Thompson [10] extended those results to obtain a linear expected time algorithm.

Bentley and Shamos have created a fast algorithm for constructing and searching an *empirical cumulative distribution function* (ECDF) [15]. An ECDF is an extension of the familiar one-dimensional cumulative distribution function. The value of the function at a point in one dimension is the number of points with smaller x coordinate. In K dimensions it is the number of points that are smaller in *all K* coordinates. Bentley and Shamos applied multi-dimensional divide-and-conquer to accomplish ECDF searching in $O(\log^K N)$ time with $O(N \log^{K-1} N)$ storage and $O(N \log^{K-1} N)$ preprocessing time. Bentley has further elaborated this in his papers on range searching [6, 9, 11].

Bentley and Saxe have characterized properties of a large class of problems called *decomposable searching problems* that include many geometric searching problems, including ECDF searching [5]. A searching problem is decomposable if the search for the relation of an object x to a set $S = A \cup B$ satisfies

$$\text{query}(x, A \cup B) = \text{query}(x, A) * \text{query}(x, B)$$

for any sets A and B such that S = A ∪ B and some binary function "*" that is computable in O(1) time. Decomposable problems have several interesting properties, one of which is that any static searching algorithm for a decomposable problem can be mechanically transformed to a dynamic searching algorithm with a loss of at most O(log N) in preprocessing time and query time. They have extended this result to a class of alternate preprocessing time, query time, and storage tradeoffs [13, 88].

1.1.5. Miscellaneous Geometric Problems

There are several topics that do not properly fit into any of the above categories. For example, we should mention that Garey, Graham, and Johnson and Papadimitriou and Steiglitz were the first to demonstrate that several geometric problems are NP-Complete [45, 74]. Also, Shamos has applied many of the techniques of computational geometry to statistics and created a new field of computational statistics [90]. ECDF searching (described above) grew from this work, and Weide's thesis [99] gives many important applications of statistical techniques to computer science problems. This includes a linear expected-time sorting algorithm for any set with an underlying distribution having a bounded probability density.

1.2. Thesis Outline

The topic of this thesis is the use of geometric transforms as tools for constructing fast geometric algorithms. The object of using a transform is, of course, to give the problem a more useful representation than it had in its original form. There is, however, no explicit rule for determining which, if any, geometric transform(s) can be applied profitably to a particular problem. Instead, we have generated *heuristics* for application of the geometric transforms. It is intended that these heuristics will help the reader use geometric transforms to solve his own problems. In the following chapters we describe a collection of geometric problems whose solutions illustrate the use of geometric transforms. The algorithms provide not only examples of the applications of the transforms but also are useful results

by themselves.

The use of transforms is not new to computer science. For example, the concept of NP-complete problems (languages) is based on polynomial time reducibility of one problem to another [1], the FFT (Fast Fourier Transform) is used for fast multiplication of polynomials [20], and many "filter" transforms are used in pattern recognition. We also encounter transforms in the solution of difference equations that arise in the analysis of algorithms (the z-transform or generating function) and solution of differential equations that arise in analysis of several types of queueing systems (Laplace transform) [61]. Yet another common example is obtaining a lower bound on the complexity of a problem X by demonstrating that an algorithm that solves X can solve a problem Y for which a lower bound is known. (Shamos' thesis [91] gives several examples of this.) Finally, we should also mention Parker's thesis [75], which explicitly addresses the application of transforms to the problems of Huffman tree construction, solution of nonlinear recurrences, and construction of permutation networks.

Chapter 2 establishes the context and direction of this thesis with a simple example - the diameter of N planar points in the L_1 and L_∞ metrics. Before this problem is discussed we first settle several issues -- model of computation, representation of the geometrical objects, measures of complexity, and, of course, a definition of diameter in the L_1 and L_∞ metrics. We present an algorithm for computing the L_∞ diameter and use a geometric transform (rotation) to transform the L_1 diameter problem to an L_∞ diameter problem. We follow the same schema followed throughout the thesis; Given a problem Y and an algorithm that solves a (related) problem X, we apply a geometric transform f that transforms problem Y to a problem of type X.

In Chapter 3 we describe the application of geometric transforms to intersection and union problems. We solve two problems in detail (the intersection of half-spaces and the union of disks) and give optimal algorithms for each. More importantly, we present several transforms and techniques in this chapter that will

be encountered many times again in succeeding chapters. The first transform introduced is a point / flat duality that transforms problems that involve flats[3] to (simpler) problems that involve points. The second new transform (inversion) converts problems that involve circles or spheres to problems of lines or planes. Inversion is typically combined with an embedding of the problem in a higher dimension to add another degree of freedom to the problem. We also introduce the convex hull (a fundamental geometrical structure) in the first of its several applications in this thesis. Finally, the techniques of Chapter 3 are tied together by deriving the point / flat duality from a limiting case of inversion combined with a convex hull and a linear transform.

In Chapter 4 we apply inversion (after embedding in a higher dimension) and convex hulls to the construction of nearest point tesselations of space. The most important such tesselation is the Voronoi diagram, which enables efficient solution of a number of geometric problems including minimum spanning tree, closest points, and Delaunay triangulation of a set of points. Shamos [89, 93, 91] applied divide-and-conquer in the plane to obtain the first O(N log N) time planar Voronoi diagram algorithm. This thesis gives a new O(N log N) time algorithm that extends straighforwardly to higher dimensions.

Chapter 5 describes two surprising applications of algorithms that search tesselations and the transforms used are the same in both cases -- the point / flat duality followed by an orthographic projection. We transform linear programming in K variables and N constraints to a problem of locating a point in a K-1-dimensional tesselation induced by N points. The problem of computing the Euclidean diameter of N points in three-space is transformed to the problem of finding all pairs of overlapping regions in two outerplanar graphs of O(N) vertices, which can be solved in O((N + K) log N) time and O(N) storage (where K is the number of pairs of antipodal vertices of the convex hull of the N points).

---

[3] A flat, also known as a hyperplane, prime, or a (K-1) flat, is a K-1-dimensional linearly closed subspace of K-space. Thus a line is a flat in the plane, a plane is a flat in three-space, etc.

In Chapter 6 we cover several miscellaneous problems that do not fall in any of the categories of the previous chapters. We describe a use of the floor function to obtain an $O(N + 1/\epsilon)$ time ϵ-approximation algorithm for the Euclidean diameter of N planar points and also demonstrate an application of gnomonic projection in an algorithm of Yuval [105] for determining if N spherical points can be fit in a hemispherical cap.

Chapter 7 summarizes the thesis and points out directions for further work. Appendix I describes the problem of choosing a good orientation for flats (before applying the point / flat duality). Appendix II gives an approach toward an $\Omega(N \log N)$ time lower bound for the Euclidean diameter of a set of N planar points, and Appendix III summarizes the geometric transforms used, their important properties, and their applications.

2. An Example: Diameter in the Plane

This chapter gives some of the flavor of geometric transforms by presenting a simple example -- the transformation of the problem of computing the L_1 diameter of a set of N planar points to the problem of computing the L_∞ diameter of another set of N planar points. But first we require some definitions and explanations along the following lines:

1. a precise problem specification,
2. a model of computation,
3. an appropriate cost measures to measure the complexity, and
4. the representation of the problem and the solution.

In the following sections we will address these issues and then construct algorithms for the L_1 and L_∞ diameters of a set of planar points.

2.1. Problem Specification

Let $S = \{ p_i = (x_i,y_i), i=1,...N \}$ be a set of N planar points. If $D(p_i,p_j)$ equals the distance (in some as-yet unspecified metric) between points p_i and p_j, then the diameter of S is

$$DIAM(S) = \max_{(i,j)} D(p_i,p_j).$$

The value of DIAM(S) depends, of course, on the metric chosen for D. The three metrics of interest in most applications are

L_1 metric: $D_1(p_i,p_j) = |x_i - x_j| + |y_i - y_j|$

L_2 (Euclidean) metric: $D_2(p_i,p_j) = ((x_i - x_j)^2 + (y_i - y_j)^2)^{1/2}$

L_∞ metric: $D_\infty(p_i,p_j) = \max(|x_i - x_j|, |y_i - y_j|)$

Let the diameters in these three metrics be denoted $DIAM_1(S)$, $DIAM_2(S)$ and $DIAM_\infty(S)$, respectively. The unit circles for these metrics are pictured in Figure

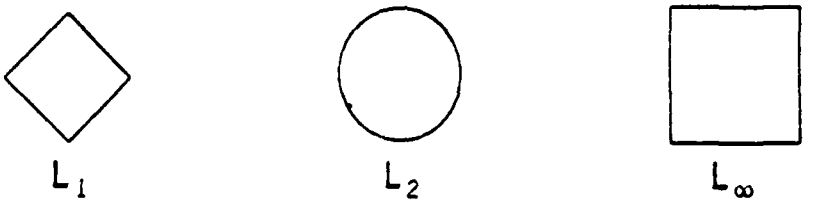


Figure 2-1: Unit circles in the L_1, L_2, and L_∞ metrics.

2-1.

The problem that we will solve is

> Given an algorithm that computes $DIAM_\infty(S)$ for any set S of N planar points, construct an algorithm that computes $DIAM_1(S)$.

The solution takes advantage of a natural isometry between the L_1 and L_∞ metrics in the plane [27]. We will cover the L_2 (Euclidean) diameter in Section 5.2.

2.2. Model of Computation

What tools are we given to compute $DIAM_1(S)$ and $DIAM_\infty(S)$? In other words, what is the model of computation? There are two (conflicting) criteria to be used in our choice: (1) how realistically the model reflects the capabilities of real machines, and (2) mathematical tractibility of the model. The *real RAM* [91] (similar to the integer RAM [1]) is a reasonable compromise for much of the work in geometric algorithms. Its capabilities are basically those of any reasonable algebraic programming language -- the four arithmetic operations (+,-,x,/), comparisons between numbers (<,≤), and indirect addressing (for convenient access to arrays and other structures). A word in a real RAM is assumed to be able to store a real number exactly; although this assumption is not entirely realistic, it is close enough for most practical applications. We often augment the arithmetic operations to include arbitrary analytic functions (trigonometric functions, exponentials, and logarithms, etc.). The floor function, on the other hand, will not be included without special comment because it is not analytic.

The floor function does seem to add power to our model of computation that is not available from analytic functions alone. Gonzalez [47] used it to find the largest gap between N (unsorted) real numbers in O(N) time and Fortune and Hopcroft [41] solved the closest-point problem in O(N log log N) worst-case time. Several fast expected-time algorithms use the floor function, including the linear expected-time closest point algorithms of Rabin [86] and Yuval [104]. Weide [99] uses it to improve his linear expected-time sorting algorithm (for all underlying distributions with bounded density) and Bentley, Weide, and Yao [18] extend Weide's result to linear expected-time Voronoi diagrams (for certain probability distributions).

2.3. Cost Measures and Complexity

Now that the model of computation has been defined we can talk about the cost or complexity of an algorithm or problem. On a real RAM each arithmetic operation, comparison, or (indirect) memory reference has an associated cost. The cost may or may not depend on the arguments for the operation, the numbers compared, or the contents of the memory referenced. The *logarithmic cost criterion* for an integer RAM [1] does assign a greater cost to manipulations (additions, comparisons, etc.) of large integers than for small integers. But for a real RAM it makes more sense to use the *uniform cost criterion* -- all operations, comparisons, and memory references have a unit cost, independent of the numbers being manipulated. We will use the uniform cost criterion throughout the thesis.

The cost of executing an algorithm is known as the *complexity* of that algorithm. The complexity of a *problem* is the minimum complexity of any possible algorithm that solves it (under the given model of computation). (The complexity of an algorithm is always an upper bound for the complexity of the problem it solves.) The complexity of an algorithm or a problem is usually expressed as a function of the *size* of the problem. The size may be the number of words of input, output, or whatever is most appropriate for the particular problem. It is often, however, inconvenient and unnecessary to obtain an exact count of all the operations,

comparison, and memory references that an algorithm makes for any particular problem size N. Knuth [62] has popularized a convenient notation for talking about asymptotic bounds on the complexity of an algorithm or problem:

> $O(f(N))$ = set of all functions $g(N)$ such that for some positive constants M and C, $|g(N)| \leq C\ f(N)$, for all $N > M$.
>
> $\Omega(f(N))$ = set of all functions $g(N)$ such that for some positive constants M and C, $g(N) \geq C\ f(N)$, for all $N > M$.
>
> $\Theta(f(N))$ = set of all functions $g(N)$ such that for some constants M, C_1, and C_2, $C_1\ g(N) \leq f(N) \leq C_2\ g(N)$, for all $N > M$.

An algorithm that solves a problem of size N in f(N) time thus proves an upper bound of $O(f(N))$ for the time complexity of the problem. If a lower bound of $\Omega(f(N))$ time is also known for that problem, then that problem has time complexity $\Theta(f(N))$. The complexity of an algorithm may alternately measure the space or storage used. The notation is the same as for time complexity, and we thus may speak of an algorithm having time complexity $O(T(N))$ and space complexity $O(S(N))$.

2.4. Representation of the Problem and Solution

How should a set S of N planar points be represented in a real RAM to enable efficient computation of $DIAM_\infty(S)$ and $DIAM_1(S)$? Many data structures would be suitable but the simplest is either an N-by-2 array or two arrays X and Y of length N. These representations are reducible to each other in linear time. Similarly, different coordinate systems for the points (X-Y vs. polar, etc.) are linear-time reducible. (The solution -- the diameter -- is simply a scalar real so its representation is not an important issue in a real RAM.) For more complicated geometrical objects such as polygons, polyhedrons, and Voronoi diagrams the issue of representation is not as easily solved, and those problems will be tackled as we come to them.

2.5. Algorithm for L_∞ Diameter of a Set of Planar Points

The L_∞ diameter of a set of planar points can now be computed fairly easily. This is because, as shown in Figure 2-1, the circle for the L_∞ metric is a rectilinearly oriented square. The L_∞ diameter is simply the diameter of the smallest rectilinearly oriented square that contains all of the points. The diameter is therefore either the difference in y coordinates of the highest and lowest points or the difference in x coordinates of the rightmost and leftmost points. Here is a pseudo-Algol description of the corresponding algorithm:

L_∞ Diameter of a Set of Planar Points

Input: integer N > 0, arrays X[1:N] and Y[1:N]
Output: L_∞ diameter of the N points
Time: O(N), Storage: Input + O(1)

```
YMin ← YMax ← Y[1];
XMin ← XMax ← X[1];
for I ← 2 thru N do
begin
    XMin ← min( X[I], XMin );
    XMax ← max( X[I], XMax );
    YMin ← min( Y[I], YMin );
    YMax ← max( Y[I], YMax );
end;
L∞Diameter ← max( XMax-XMin, YMax-YMin )
```

The O(N) time complexity of the above algorithm is optimal to within a constant factor because the algorithm must read all of its N inputs to ensure a correct answer. There is, however, room for improvement; for instance, the computation of max and min can be done in less than 3/4 as many comparisons as are taken above [76]. Note that the storage required is actually O(1) rather than O(N) because no computation involves more than the Ith element of X and Y at any given time. The values in X and Y can therefore be read from a tape rather than stored in arrays.

2.6. Algorithm for L_1 Diameter of a Set of Planar Points

In this section we will construct an algorithm for the L_1 diameter of a set of N points. We could start from scratch, but since the L_∞ diameter algorithm is already available, it would be nice to be able to make the L_1 diameter problem look like an L_∞ diameter problem, that is, *transform* it to an L_∞ diameter problem. Fortunately, this can be done, and the clue is in Figure 2-1. The circle for the L_1 metric can be made to look like the circle for the L_∞ metric if it is simply rotated 45 degrees (and multiplied by a scale factor of $2^{1/2}$.) This leads us to the intuitive algorithm below:

L_1 Diameter of a Set of Planar Points

```
Input: integer N > 0, arrays X[1:N] and Y[1:N]
Output: L1 Diameter
Time: O(N), Storage: O(N)
        ! Rotate the points 45 degrees;
            for I ← 1 thru N do
            begin
                X' ← (X[I] + Y[I]) / 2^1/2;
                Y' ← (-X[I] + Y[I]) / 2^1/2;
                X[I] ← X';   Y[I] ← Y';
            end;
        ! Compute L∞ Diameter and scale by 2^1/2;
            L1 Diameter ← DIAM∞(X,Y,N) · 2^1/2;
```

The hard part is proving that this algorithm is correct. Since the diameter is simply the maximum interpoint distance, it will be sufficient to show that computing the L_1 distance by the definition in Section 2.1 is equivalent to the computation in the algorithm above. Let $p_i = (x_i,y_i)$ and $p_i' = p_i$ rotated $\pi/4$ radians about the origin. (The rotation can be clockwise or counterclockwise, as long as it is the same in each case. In the algorithm above, we use the formulas $x' = (x + y)\cos(\pi/4)$ and $y' = (-x + y)\cos(\pi/4)$ to rotate the points $\pi/4$ radians clockwise.) The two methods for computing the L_1 distance between p_i and p_j are:

1. (Definition) $L_1\text{Distance}(p_i,p_j) = |x_i - x_j| + |y_i - y_j|$, and

2. (Algorithm) $L_1\text{Distance}(p_i,p_j) = 2^{1/2} \cdot L_\infty\text{Distance}(p_i',p_j')$.

We prove this by reducing the second (algorithm) formula to the first (definition) formula:

$$\begin{aligned}\sqrt{2} \cdot L_\infty\text{Distance}(p_i',p_j') &= \sqrt{2} \cdot \max(|x_i' - x_j'|, |y_i' - y_j'|) \\ &= \sqrt{2} \cdot \max\left[\left|\frac{x_i+y_i}{\sqrt{2}} - \frac{x_j+y_j}{\sqrt{2}}\right|, \left|\frac{-x_i+y_i}{\sqrt{2}} - \frac{-x_j+y_j}{\sqrt{2}}\right|\right] \\ &= \max(|(x_i-x_j) + (y_i-y_j)|, |-(x_i-x_j) + (y_i-y_j)|) \\ &= L_1\text{Distance}(p_i,p_j)\end{aligned}$$

There are four possible cases to satisfy:

1. $x_i - x_j < 0$, $y_i - y_j < 0$,

2. $x_i - x_j \geq 0$, $y_i - y_j \geq 0$,

3. $x_i - x_j \geq 0$, $y_i - y_j < 0$, and

4. $x_i - x_j < 0$, $y_i - y_j \geq 0$.

In each of these cases the identity holds. The algorithm for computing the L_1 diameter is therefore correct.

2.7. Principles Covered

In this chapter we solved a simple geometric problem -- computing the L_1 diameter of a set of planar points -- and demonstrated the use of a geometric transform. Several principles have been presented that will be encountered repeatedly in this thesis: precise specification of the problem, choice of a model of computation, cost measures and analysis of the complexity, representation of the problem and its solution, and, of course, the use of a geometric transform. The choice of the transform (rotation) in the example of this chapter may still seem like something pulled out of a hat. Yet there is a method to it, as the following chapters will demonstrate.

3. Intersection and Union Problems

In this chapter we introduce two important techniques -- the use of a point / flat duality and the combined use of inversion with embedding in a higher dimension -- and apply them to two geometric intersection problems. The first problem is the intersection of N (UPPER) half-spaces and the second is the union of (the interiors of) N circles. For both of these problems we develop algorithms that are optimal (within a constant factor). Finally, the last section of this chapter shows that the techniques that we used for these two problems are actually more closely related than they appear to be.

3.1. Intersection of Half-Spaces

In this section we analyze the problem of constructing the intersection of a set of N (UPPER) half-spaces. The first topic that we cover is the *representation* of half-spaces and their common intersection in a computer. Given this representation we then prove upper and lower bounds on the complexity of constructing the intersection in two, three, and higher dimensions. We conclude with fast expected-time algorithms and some open problems.

The reader should carry away three important tools for the construction of geometric algorithms:

- A point / flat duality that is applicable to a number of problems in this thesis. It is used for transforming (formidable) problems that involve flats to (simpler) problems that involve points.

- A fast algorithm for intersection of (UPPER) half-spaces. (An algorithm for intersection of half-spaces is useful for linear programming in two or three variables [91], intersection of convex polyhedra, and as a tool for solving other geometric problems.)

- The first of several important uses of the convex hull of a point set.

We will use these tools many times in succeeding chapters.

3.1.1. Representation of Half-Spaces and Their Intersection

The first requirement of any representation of a geometric object is that it contain all of the necessary information to describe the object, and the second requirement is that it provide the information efficiently (both for encoding and decoding). We shall first describe such a representation for the two-dimensional case (half-planes and intersections of half-planes) and then extend our representation to an arbitrary number of dimensions. The details of our representation can be easily modified to a number of forms that can be reduced to one another in constant time for a single object.

We represent a half-plane by the line bounding the half-plane and a single bit to indicate which side of the line the half-plane is on. There are many ways to represent the boundary line, but we will use the slope-intercept form with the understanding that vertical (or near-vertical) lines will require exceptional handling (Appendix I).[4] The half-plane

$$y \leq ax + b$$

can thus be represented as (a,b,0) and the half-plane

$$y \geq ax + b$$

can be represented as (a,b,1). In a computer these may be three (scalar) variables or, if there are many half-planes, three elements of an array.

The representation of the intersection of N half-planes is more interesting. Certainly one (cheap) method is to represent the N half-planes as described above and include a scalar flag INT that indicates that the intersection is intended. This has the advantage of representing the intersection fast (in linear time) but the disadvantage that it doesn't help us quickly answer important questions about the intersection, such as "Is the intersection empty?". Another possibility is to append to each of the N half-planes a flag that indicates whether or not part of the

---

[4] Preparata and Muller [84] use a homogeneous coordinate representation, which treats all coordinates uniformly.

boundary of the half-plane is also a boundary of the intersection. (If the boundary of half-plane i does not meet the intersection of the N half-planes, then half-plane i is redundant.) This representation enables us to answer quickly whether or not the intersection is empty but in the worst case it does not enable a faster solution to questions of the form "Is point P inside the intersection?". To answer such questions quickly we must store the nonredundant half-planes in sorted order. Since the intersection of N half-planes is a (possibly empty) convex polygonally bounded region with at most N edges, the representation that we will use is the quadruple

$$(V, M[1:V], B[1:V], F[1:V]).$$

Here V is the number of edges in the intersection, M and B are the slope and intercept, respectively, of the lines determined by the V edges, sorted in counterclockwise order, and F is a bit vector that allows us to quickly distinguish the inside from the outside of the intersection.

The representation of N K-dimensional half-spaces is a simple extension of the two-dimensional case. If the half-space is

$$x_k \leq \sum_{i=1}^{k-1} a_i x_i + a_k$$

then the representation is simply

$$(a_1, a_2, \ldots a_{K-1}, a_K, 0).$$

Similarly, if the "$\leq$" is replaced by a "$\geq$" in the equation above, then the "0" will be replaced by a "1" in the representation. In a computer, we can represent N K-dimensional half-spaces in one large array A[1:N,0:K] where the "side" bits are stored in the entries A[i,0].

The intersection of N K-dimensional half-spaces is more difficult to represent. This is because the total number of vertices, edges, faces, hyperfaces, etc. grows exponentially with K. (If we choose to record only the half-spaces with flags that indicate for each half-space whether or not it is redundant, then only linear storage

is required. Unfortunately, as we mentioned for the two-dimensional case, we would not then be able to answer quickly questions of the form "Is point P in the intersection?".) We must first establish some terminology. Let a vertex be called a 0-face, an edge a 1-face, and, in general, a j-dimensional piece of the intersection be a j-face. We will represent the intersection of K-dimensional faces by enumerating the j-faces (for $0 \leq j \leq K-1$) and recording how they are interconnected. The representation of the intersection of N K-dimensional half-spaces is the septuple

(H, F, V, Connect1, Connect1Ptr, Connect2, Connect2Ptr),

where

H[1:h,1:K] represents the set of h flats determined by the K-1-faces of the intersection,

F[1:h] is a bit-vector of flags that enable us to distinguish the inside from the outside of the intersection.

V[1:v,1:K] is the set of v vertices determined by the N half-spaces,

Connect1 is a table of l-faces used by Connect1Ptr,

Connect1Ptr[l,J] is the subscript of Connect1 for the first l+1-face that the Jth l-face bounds,

Connect2 is a table of l-faces used by Connect2Ptr,

Connect2Ptr[l,J] is the subscript of Connect2 for the first l-1-face that determine the Jth l-face, in counterclockwise order,

Note that the four "Connect" arrays are jagged arrays rather than rectangular arrays. They are also redundant, for ease of use.

3.1.2. Lower Bound

We prove an $\Omega(N \log N)$ time lower bound for the intersection of N half-planes by demonstrating that an algorithm that intersects half-planes can be used to sort. (The lower bound applies for all half-spaces of dimension $K \geq 2$ because half-planes are just a special case of K-dimensional half-spaces.) Our construction follows that of Shamos [91].

Theorem 1: The intersection of N half-planes requires $\Omega(N \log N)$ time in the worst case.

Proof: Given N real numbers a_i, i=1, ... N we construct N half-planes h_i by

$$h_i: y \geq a_i x + (a_i/2)^2.$$

These half-planes h_i contain the origin and are bounded by lines that have slope a_i and are tangent to the parabola $y = x^2$. The intersection of the h_i is a convex polygonal region whose edges are sorted by slope. We simply read off the slopes of these edges to obtain the a_i in sorted order.

The proof of the lower bound for intersection of half-planes requires a lower bound for sorting under a model of computation that can support the operations used in our construction above. This has been provided by Friedman [43], who proved an $\Omega(N \log N)$ time lower bound for sorting under a model of computation that allows analytic functions of the input. Since our construction requires only analytic functions, the $\Omega(N \log N)$ lower bound for sorting applies also to the intersection of half-planes.

3.1.3. Intersection of Half-Planes

Shamos [89] and Shamos and Hoey [94] show that the intersection of N half-planes has time complexity $\Theta(N \log N)$. Their algorithm for constructing the intersection in $O(N \log N)$ time recursively applies their linear-time algorithm for intersecting two convex N-gons. The algorithm that we describe below, on the other hand, is based on a geometric transform (point / flat duality) that maps the problem of intersecting half-planes to two problems of constructing the convex hull of a planar set of points (and a simple intersection problem.) Furthermore, it extends to higher dimensions (unlike Shamos and Hoey's algorithm). We next describe the decomposition of the two-dimensional problem into three subproblems. The following sections characterize redundant half-planes, introduce the point / flat duality transform and then apply the transform to the intersection of half-planes.

3.1.3.1. The Two-Dimensional Problem

In Figure 3-1 we illustrate the intersection of N half-planes. The intersection itself is indicated by the shaded region. We partition the half-planes into two sets, UPPER and LOWER. A half-plane is in set UPPER if the line at its boundary is above the rest of the half-plane. Similarly, a half-plane is in the set LOWER if the line at its boundary is below the rest of the half-plane. (If any boundary lines are vertical, then we rotate all N half-planes a small angle.) The reason that we produce this partition is that the transform (to be described) actually applies only to lines, not half-planes. Since each line may be associated with two half-planes, we partition the set of half-planes into two parts so that the half-plane determined by a line will not be ambiguous. Our partition of the half-planes also enables us to divide the problem of intersecting the half-planes into three parts:

1. Construction of U, the intersection of the UPPER half-planes,
2. Construction of L, the intersection of the LOWER half-planes, and
3. The intersection of U and L.

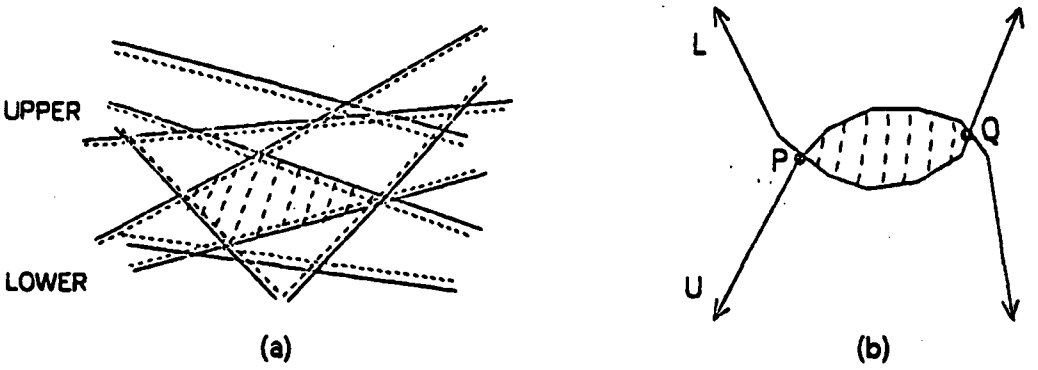


Figure 3-1: (a)Intersection of N half-planes., (b) Intersection of regions U and L.

As shown in Figure 3-1b, part (3) is relatively easy. If U and L have O(N) vertices, then the intersection (shaded region) can be constructed in O(N) time. We

describe the algorithm in detail as Algorithm IntersectChains below.

Algorithm IntersectChains

Input: Intersections of half-planes U = (N1, UM[1:N1], UB[1:N1]) and L = (N2, LM[1:N2], LB[1:N2]) where N1 and N2 are integers such that N = N1 + N2 and UM, UB, LM, and LB are the slopes and intercepts of the lines determined by the edges of U and L.[5] The edges are sorted in counterclockwise order:

$$UM[1] < UM[2] < \ldots < UM[N1]$$
$$LM[1] < LM[2] < \ldots < LM[N2]$$

Output: Integer E (number of vertices of the intersection), arrays M[1:E], B[1:E] (slopes and intercepts of the E edges), bit vector F[1:E] to distinguish the inside from the outside of the half-planes.

Time: O(N), Space: O(N).

1. Scan U and L (vectors UM, UB, LM, and LB) from left to right until two segments intersect at a point P. (If no segments intersect then the intersection of U and L is empty.) The scan can be done in O(N) time in a manner similar to the O(N) time merge in the merge sort algorithm.

2. Scan U and L from right to left until two segments intersect at a point Q.

3. If P ≠ Q, then return (in vectors M and B) the concatenation of the chains of line segments of U and L between points P and Q.

4. If P = Q, then the intersection is unbounded (or just the point P = Q). In the case of an unbounded intersection we must determine whether to return the chains to the left of P or the chains to the right of P. This can be determined by comparing the slopes of the rays bounding the left and right sides of U and L. If the slope of the left ray of U is less than the slope of the left ray of L, then return the chains to the left of P. Otherwise, return the chains to the right of P.

---

[5] Since U is an intersection of UPPER half-planes and L is an intersection of LOWER half-planes it is not necessary to include bit vectors indicating inside vs. outside of the half-planes.

We have just seen how to construct efficiently the intersection of N half-planes, given U and L, the intersections of the UPPER and the LOWER half-planes. Now we must design a fast algorithm for constructing U and L. Since the construction of L is so similar to the construction of U, we will describe only the construction of U.

3.1.3.2. Redundant Half-Planes

Assume that the N half-planes are all UPPER half-planes. Some of these half-planes, such as half-plane k in Figures 3-2 (a) and (b), do not bound any side of the region of intersection. It would be nice if we could find all such half-planes and throw them away since they do not contribute to the final result. Once that is done, we can find the intersection of the UPPER half-planes rather easily. As we can see in Figure 3-1 (b), the slopes of the sides of the chain U are monotonic decreasing as we travel from left to right. Given the lines determined by the sides, we need only sort the lines by slope to determine the order in which they intersect to form the sides. Since the sort costs only O(N log N) time, we can construct the intersection of the UPPER half-planes in O(N log N) time, once the redundant half-planes have been eliminated.

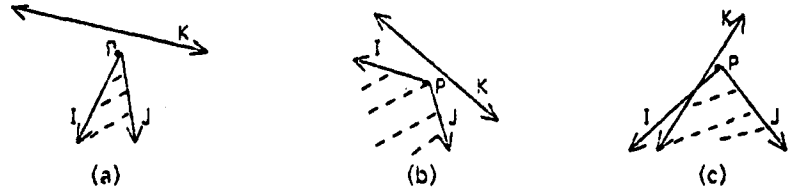


Figure 3-2: (a) & (b) - k is redundant, (c) - k is nonredundant.

How do we determine which half-planes are redundant and which are not? There are two conditions that we need to check.

Theorem 2: An UPPER half-plane k is redundant with respect to UPPER half-planes i and j iff

(A) Line k is above the point P where lines i and j meet, and

(B) The slope of line k lies between the slopes of lines i and j.

Proof: The half-spaces i, j, and k are

$$y \le a_i x + b_i, \quad y \le a_j x + b_j, \quad \text{and} \quad y \le a_k x + b_k$$

and *lines* i and j meet at point

$$P = (P_x, P_y) = (-(b_i - b_j)/(a_i - a_j), (a_i b_j - a_j b_i)/(a_i - a_j)).$$

(See Figure 3-2.) Half-plane k is redundant with respect to half-planes i and j iff line k lies above the two rays r_i and r_j originating at point P and defining the boundaries of the intersection of half-spaces i and j. There are six possible cases to consider:

$$0 < a_i < a_j, \quad 0 < a_j < a_i, \quad a_i < 0 < a_j$$
$$a_j < 0 < a_i, \quad a_i < a_j < 0, \quad \text{and } a_j < a_i < 0.$$

Since many of these are equivalent, we need to prove only the two cases

$$0 < a_j < a_i \quad \text{and} \quad a_j < 0 < a_i.$$

Case (1) $0 < a_j < a_i$: The ray r_i points (downward) in the direction $(1,-a_i)$ and the ray r_j points (upward) in the direction $(1,a_j)$. Line k lies above r_i and r_j iff

$$P_y - a_i u \le a_k (P_x - u) + b_k, \forall u \ge 0 \text{ and} \tag{1}$$

$$P_y + a_j u \le a_k (P_x + u) + b_k, \forall u \ge 0. \tag{2}$$

Letting $u = 0$ in either (1) or (2), we see that line k lies above point P, satisfying condition (A) above. To prove condition (B), that a_k lies between a_i and a_j, we divide by u in (1) and (2) and then take the limit as $u \to \infty$, obtaining $a_i \ge a_k$ and $a_j \le a_k$, respectively. Conversely, if (A) is satisfied ($P_y \le a_k P_x + b_k$) and (B) is satisfied ($0 < a_j < a_k < a_i$), then inequalities (1) and (2) immediately follow.

Case (2) $a_j < 0 < a_i$: The ray r_i points (downward) in the direction $(1,-a_i)$ and the ray r_j points (downward) in the direction $(1,-a_j)$. Line k lies above r_i and r_j iff inequalities (1) and (2) hold. The proof is very similar to the proof for Case (1). □

How fast can we determine (non)redundancy for each of the N UPPER half-planes? Certainly one approach is to test all pairs of half-planes i and j for each half-plane k. That costs $O(N^3)$ time, though, which leaves much room for

improvement. In the next section we show an entirely different way of looking at this problem that solves it in a natural and efficient way.

3.1.3.3. A Point / Line Duality Transform

In this section we present a transform that exploits a natural duality between points and lines in the plane. A line in slope-intercept form ($y = ax + b$) is uniquely identified by the pair (a,b). (This transform will not work for vertical lines.) We thus have a natural mapping from lines to points. We can also map points to lines. For an arbitrary point (x,y) the set of all lines (in slope-intercept form) that pass through that point can be represented by the set $\{ (a,b) \mid y = ax + b \}$. This transforms a point (x,y) to a line $b = -xa + y$. Points thus transform to lines and lines transform to points by the formulas

$$y = ax + b \quad \rightarrow \quad (a,b), \text{ and}$$

$$(x,y) \quad \rightarrow \quad b = -xa + y.$$

This duality is illustrated in Figure 3-3.

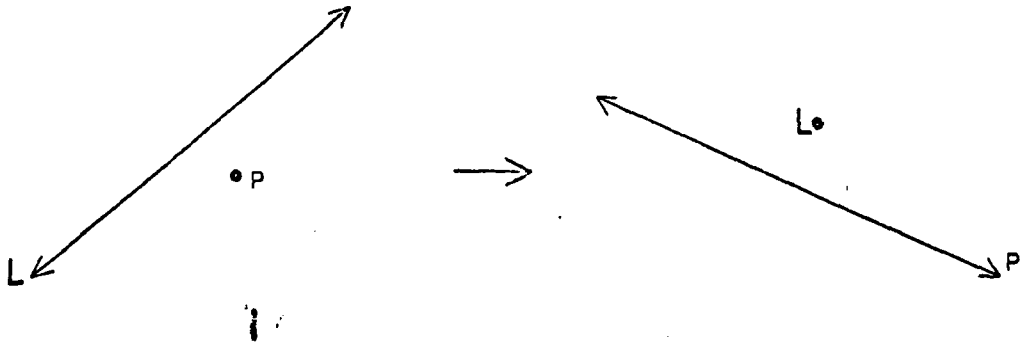


Figure 3-3: Point / Line Duality

This transform has an interesting property: Distances *in the y-coordinate* between points and lines are preserved.[6] The difference in y coordinate between

---

[6]The restriction *in the y coordinate* is important because it can be shown to be impossible to preserve the *Euclidean* distance between a point and a line under a duality transform [02].

point (c,d) and line $y = ex + f$ is $d - (ec + f)$. The difference in the transforms $b = -ca + d$ and (e,f) is $(-ce + d) - f$,, which is the same. It follows from this that incidence is preserved.[7] If point (c,d) is on line $y = ex + f$, then it holds also for their transforms -- point (c,f) is on line $b = -ca + d$. Note further that not only is the magnitude of the distance (in the y-coordinate) preserved but also its sign. Thus, above/belowness is preserved. If (c,d) is above (below) line $y = ex + f$, then the transform of (c,d) is above (below) the transform of $y = ex + f$.

There is another property of the transform that we should mention. The transform is not involutory, but composition of it *four* times produces the following:

$$(x,y) \rightarrow b = -xa + y \rightarrow (-x,y) \rightarrow b = xa + y \rightarrow (x,y)$$

Only a slight change is required to make the transform its own inverse: express lines in the form $y + ax + b = 0$ rather than $y = ax + b$. Then it is true that $y + ax + b = 0 \Leftrightarrow (a,b)$. But this has the unfortunate side effect that above/belowness between points and lines is not preserved; it is reversed. If point (c,d) is above line $y + ex + f = 0$ then the transform of (c,d) will be *below* the transform of line $y + ex + f = 0$.

3.1.3.4. Application of the Transform to the Two-dimensional Problem

We now show how the transform enables us to intersect the UPPER (or LOWER) half-planes fast. More specifically, the transform enables an efficient mechanism for eliminating the "redundant" half-planes. Recall the two conditions for redundancy of an UPPER half-plane: a half-plane k is redundant iff there exist half-planes i and j such that (1) line k is above the point P where lines i and j intersect, and (2) the slope of k is between the slopes of lines i and j. In the ab plane there is a corresponding interpretation.

In Figure 3-4, line k is above point P in the xy plane. This is transformed to a *point* k that is above *line* P in the ab plane. (Above/belowness between lines and

---

[7] There are other duality transforms that preserve incidence, such as Plucker's transform [91].

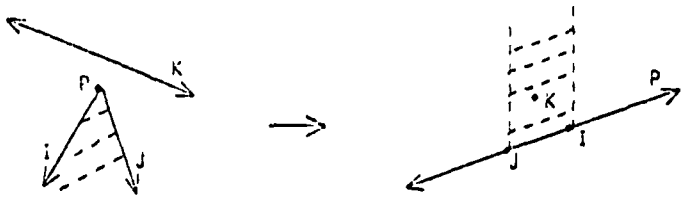


Figure 3-4: Transform of a redundant half-plane.

points is preserved by the transform.) The slope of a line in the xy plane is the a coordinate of the corresponding point in the ab plane. Line k thus has a slope between the slopes of lines i and j and transforms to a point k with a coordinate between the a coordinates of points i and j in the ab plane. Figure 3-5 shows the result of applying the transform to a set of N UPPER half-planes. A point in the ab plane corresponds to a redundant half-plane iff it is directly above one (or more) of the line segments determined by the N - 1 other points.

Figure 3-5: Transform of N UPPER half-planes.

Theorem 3: Given a set of N UPPER half-planes of the form

$$y \leq a_i x + b_i,$$

and a mapping

$$y \leq a_i x + b_i \quad \rightarrow \quad (a_i, b_i),$$

the nonredundant half-planes correspond to those points on the bottom part of the convex hull of the N points

$$(a_i, b_i).$$

Proof: The proof is in two parts: (1) a point on the bottom part of the convex hull corresponds to a nonredundant half-plane, and (2) a nonredundant half-plane transforms to a point on the bottom part of the

convex hull of the N points.

1. Let P be a point on the bottom part of the convex hull of the N points in the ab plane. P does not lie above any segment connecting two of the N points because P would then not be on the bottom part of the hull. It follows that P can not be redundant.

2. If a half-plane P is nonredundant, then it transforms to a point P that does not lie above any segment connecting two of the N - 1 other points in the ab plane. P must be on the bottom part of the convex hull because otherwise it would lie above such a segment.

□

We have reduced the problem of intersecting N UPPER half-planes to the problem of constructing the (bottom part of the) convex hull of N points. The convex hull of N points in the plane can be constructed in O(N log N) time [48]. This leaves only the detail of separating the top from the bottom part of the hull. To do that, we find the leftmost point of the hull in O(N) time and then traverse the hull on the *bottom* side until the rightmost point is reached.

Theorem 4: The intersection of N half-planes can be constructed in O(N log N) time.

Proof: We have broken the problem of intersecting N half-planes into three parts. Part (1), the intersection of the UPPER half-planes, has been shown to cost only O(N log N) time. Part (2), the intersection of the LOWER half-planes, is equivalent to part (1) so it can also be done in O(N log N) time. Part (3), the intersection of the results of parts (1) and (2), costs only O(N) time. The intersection of N half-planes can thus be solved in O(N log N) time. □

It is interesting to note that we can also use an intersection of half-planes algorithm to produce a convex hull of points algorithm. We first transform all N points to UPPER half-planes by the formula

$$(x,y) \rightarrow b = xa + y$$

and intersect the half-planes. We transform back to obtain the lower part of the hull. Then we transform all N points to LOWER half-planes and intersect the half-planes and transform back for the upper part of the hull. Merging the upper and lower parts is trivial, since the leftmost and rightmost points will be in each one. The total time required is O(N log N).

3.1.4. Intersection of Three-Dimensional Half-Spaces

The technique that we just used in two dimensions can be extended to the intersection of N UPPER (or N LOWER) three-dimensional half-spaces. (Zolnowsky [106] and Preparata and Muller [84] describe all the details require to solve the general problem of intersecting three-dimensional half-spaces.) The following sections extend the concept of redundant half-space to three dimensions and apply the (three-dimensional) point / flat duality to construct the intersection of N UPPER half-spaces in O(N log N) time.

3.1.4.1. Redundancy in Three Dimensions

The algorithm for constructing U in three dimensions is analogous to the algorithm for the two-dimensional case. We first transform the N UPPER half-spaces to N points in abc space and construct the (bottom part of the) convex hull of the points in O(N log N) time. The points above the bottom part of the convex hull correspond to "redundant" half-spaces and can be discarded. To form U we simply apply an inverse transform to the bottom part of the convex hull. We will now describe this procedure in detail.

Assume that there are N UPPER half-spaces. Some of these half-spaces contribute to the intersection U and some are "redundant," such as half-space M in Figure 3-6. For the plane there are two simple conditions for redundancy of a half-plane. In three dimensions there are two analogous conditions for a half-space. The first condition for redundancy of a half-space M is that plane M be above the point P where planes I, J, and K intersect, as in Figure 3-6. The second condition, the "betweenness of slopes" condition, is more complicated to express. The

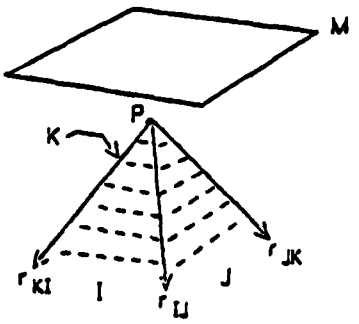


Figure 3-6: A redundant half-space.

purpose of the "betweenness of slopes" condition is to ensure that a plane above the point P cannot drop down fast enough to enter the region below planes I, J, and K. We will now derive an algebraic description of these two conditions.

Let plane M be written

$$z = a_M x + b_M y + c_M$$

and let planes I, J, and K be written

$$z = a_I x + b_I y + c_I,$$
$$z = a_J x + b_J y + c_J, \text{ and}$$
$$z = a_K x + b_K y + c_K.$$

The point $P = (P_x, P_y, P_z)$ where planes I, J, and K meet is defined by

$$\begin{bmatrix} -a_I & -b_I & 1 \\ -a_J & -b_J & 1 \\ -a_K & -b_K & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} = \begin{bmatrix} c_I \\ c_J \\ c_K \end{bmatrix}. \qquad (3)$$

The three rays r_{IJ}, r_{JK}, and r_{KI} can be expressed as vectors originating at point P. The directions of these three vectors are obtained by computing the cross products of the normals to planes I, J, and K. For example,

$$r_{IJ} = (\alpha_{IJ}, \beta_{IJ}, \gamma_{IJ}) = \begin{vmatrix} \bar{I} & \bar{J} & \bar{K} \\ a_I & b_I & -1 \\ a_J & b_J & -1 \end{vmatrix} \times \begin{vmatrix} a_I & b_I & -1 \\ a_J & b_J & -1 \\ a_K & b_K & -1 \end{vmatrix}$$

where $\bar{I}$, $\bar{J}$, and $\bar{K}$ are unit vectors along the x, y, and z axes, respectively. (Similarly for r_{JK} and r_{KI}.) We may now express the two conditions for redundancy of a half-space.

Theorem 5: Let UPPER half-spaces I, J, K, and M, point P, and rays r_{IJ}, r_{JK}, and r_{KI} be as given above. Half-space M is redundant with respect to half-spaces I, J, and K iff

$$a_M P_x + b_M P_y + c_M \geq P_z \tag{4}$$

and

$$\begin{aligned} a_M\alpha_{IJ} + b_M\beta_{IJ} &\geq \gamma_{IJ} \\ a_M\alpha_{JK} + b_M\beta_{JK} &\geq \gamma_{JK} \\ a_M\alpha_{KI} + b_M\beta_{KI} &\geq \gamma_{KI}. \end{aligned} \tag{5}$$

Proof: M is redundant iff it lies above all points of the three rays r_{IJ}, r_{JK}, and r_{KI}. But M lies above these three rays iff

$$\begin{aligned} a_M (P_x + u\alpha_{IJ}) + b_M (P_y + u\beta_{IJ}) + c_M &\geq (P_z + u\gamma_{IJ}), \forall u>0, \\ a_M (P_x + u\alpha_{JK}) + b_M (P_y + u\beta_{JK}) + c_M &\geq (P_z + u\gamma_{JK}), \forall u>0, \text{ and} \\ a_M (P_x + u\alpha_{KI}) + b_M (P_y + u\beta_{KI}) + c_M &> (P_z + u\gamma_{KI}), \forall u>0 \end{aligned}$$

Letting u = 0 we obtain the first condition (inequality (4)) for redundancy of M

$$a_M P_x + b_M P_y + c_M \geq P_z$$

and dividing by u and taking the limit as $u \to \infty$ we obtain the other three conditions (5). Conversely, if the four inequalities of (4) and (5) are satisfied, then M must lie above all points of the three rays r_{IJ}, r_{JK}, and r_{KI}. We prove this by simply multiplying both sides of the last three inequalities (5) by a $u > 0$ and adding the result to the first inequality (4).

□.

3.1.4.2. Application of the Transform to the Intersection of Half-Spaces

The transform that we use in three dimensions is a straightforward extension of the two-dimensional transform; planes transform to points and points transform to planes. The formulas for the transform are:[8]

$$\begin{aligned} z = ax + by + c &\rightarrow (a,b,c), \text{ and} \\ (x,y,z) &\rightarrow c = -xa + -yb + z. \end{aligned} \tag{6}$$

The distance between a point and a plane *in the z coordinate* is preserved by this transform and, most importantly, the sense of above/belowness (in the z coordinate) between points and planes is also preserved.

Theorem 6: Let the UPPER half-spaces I, J, K, and M transform to points P_I, P_J, P_K, and P_M by the transform (6). Half-space M is redundant with respect to half-spaces I, J, and K iff point P_M is directly above some point in (or on) the triangle determined by points P_I, P_J, and P_K.

Proof: Plane M ($z = a_M x + b_M y + c_M$) of Figure 3-6 transforms to the point (a_M,b_M,c_M) in abc space, and planes I, J, and K transform to the points

$$P_I = (a_I,b_I,c_I), \quad P_J = (a_J,b_J,c_J), \quad \text{and} \quad P_K = (a_K,b_K,c_K).$$

The inequality (4) (of the previous section) for redundancy of half-space M (point (a_M,b_M,c_M)) can be rewritten

$$c_M \geq (-P_x)a_M + (-P_y)b_M + P_z.$$

The interpretation in abc space is that point (a_M,b_M,c_M) lies above the plane

$$c = (-P_x)a + (-P_y)b + P_z.$$

By inspection of Equation (3) we also see that this plane is determined by the three points P_I, P_J, and P_K. The three "betweenness of slopes" conditions (5) define three vertical planes, each of which passes through two of the three points P_I, P_J, and P_K. Since half-spaces I, J, and K are all UPPER half-spaces, the set of redundant points P_M is a bounded

---

[8] Dantzig [28] uses the above transform in the context of linear programming and Huffman [53] uses an almost identical transform for an analysis of polyhedral scenes. Kanade [57] uses this transform for what he calls the "origami world" of three-dimensional figures.

region of abc space. The intersection of the three half-spaces of conditions (5) and the half-space (4) is therefore the region in, on, or above the triangle determined by points P_I, P_J, and P_K. □

Corollary: Let the N UPPER half-spaces H_1, H_2, . . . H_N transform to points P_1, P_2, . . . P_N by Equation (6). A half-space H_I is redundant iff point P_I does *not* lie on the bottom part of the convex hull of points P_1, P_2, . . . P_N.

We can construct the (bottom part of the) convex hull of N three-dimensional points in O(N log N) time. We do this by augmenting the algorithm of Preparata and Hong [83] for constructing the (entire) convex hull with a mechanism for separating the bottom part of the hull from the top. One way to do this is to maintain with each face F of the convex hull a vector perpendicular to F that points toward the inside (as opposed to the outside) of the hull. The bottom faces of the hull are those faces whose vectors point upward and the vectors for the top faces point downward. (Note that there will be some *vertices* in both the top and bottom parts of the hull. These are the vertices that bound both top and bottom faces.) The bottom and top parts of the convex hull are therefore separable in O(N) time, once the entire convex hull is constructed in O(N log N) time.

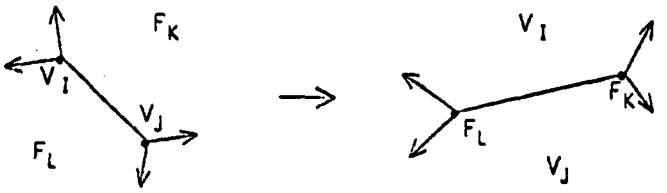


Figure 3-7: Transform of a convex hull.

To find the intersection of the UPPER half-spaces, we must transform the bottom part of the convex hull to xyz space. The O(N) vertices transform, of course, to planes. But there is much more information than that in the convex hull. For instance, defining each face of the hull there are three coplanar vertices I, J, and K.

[9] The plane that these vertices define is transformed to a point in xyz space. This point is where *planes* I, J, and K (of xyz space) intersect. (This follows from the fact that the transform preserves incidence between points and planes.) Also, as illustrated in Figure 3-7, if faces F_K and F_L of the (bottom part of the) convex hull share an edge V_IV_J, then in xyz space *faces* V_I and V_J share an edge F_KF_L. In fact, even the unbounded faces of U in xyz space can be obtained from the transform. These faces of U correspond to vertices at the boundary between the top and bottom parts of the convex hull in abc space. Little computation is thus required to construct U after the transform from abc space since all the faces, vertices, edges, etc. are directly obtainable from the transform. We have

Theorem 7: The intersection of N UPPER three-dimensional half-spaces can be constructed in O(N log N) time.

Proof: Since the transform costs only O(N) time, the total time to construct U is dominated by the time to construct the convex hull in abc space, which is O(N log N) time. □

3.1.5. Intersecting Half-spaces in Four or More Dimensions

Suppose that in K dimensions we can construct the convex hull of N points in H(N,K) time; we can then construct the intersection of N UPPER K-dimensional half-spaces in O(H(N,K)) time. The algorithm is a straightforward extension of the one that we used in the three-dimensional case. We first transform the N half-spaces to N points in K-space. Then we construct the convex hull of these N points in H(N,K) time. Then we partition the top from the bottom part of the hull and transform the bottom part back to obtain the intersection of the N UPPER half-spaces. of the hull, and transform the bottom (top) part of the hull back.

In this section we present an algebraic description of the components of the K-dimensional algorithm. The first step is to describe algebraically the conditions under which an UPPER half-space M is redundant with respect to a set S of K UPPER

---

[9] Assuming that the vertices are in general position. If not, then there may be four or more coplanar vertices.

half-spaces. We then present a general duality transform for K-space and interpret the conditions for redundancy in the transform space. Finally, we characterize redundancy among N UPPER half-spaces in terms of the convex hull of the N points to which they transform.

3.1.5.1. Algebraic Description of Redundancy

We must first introduce some terminology. Let a "j-face" of a K-dimensional polytope be denoted as follows: a vertex is a 0-face, an edge (a line segment) is a 1-face, etc. Let S be a set of K UPPER K-dimensional half-spaces defined by

$$x_K \leq \sum_{j=1}^{K-1} a_{ij}x_j + a_{iK}, \quad i=1, \ldots, K$$

and let M be an UPPER K-dimensional half-space defined by

$$x_K \leq \sum_{j=1}^{K-1} a_{Mj}x_j + a_{MK}.$$

Let the matrices B and C be defined by

$$B = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,K-1} & -1 \\ a_{21} & a_{22} & \cdots & a_{2,K-1} & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{K1} & a_{K2} & \cdots & a_{K,K-1} & -1 \end{bmatrix} \quad \text{and} \quad C = -\begin{bmatrix} a_{1K} \\ a_{2K} \\ \cdot \\ \cdot \\ \cdot \\ a_{KK} \end{bmatrix}.$$

The flats bounding the K half-spaces of S meet at a point $P = (P_1, P_2, \ldots, P_K)$ defined by

$$BP' = C. \tag{7}$$

In three dimensions the planes I, J, and K determine three rays r_{IJ}, r_{JK}, and r_{KI} originating at point P. In K dimensions we have K such rays, denoted $r_1, r_2, \ldots r_K$ where r_i is the ray determined by the K-1 elements of S - {i}. If $(\alpha_{i1}, \alpha_{i2}, \ldots \alpha_{iK})$ is a vector pointing in the same direction as ray i, then we may write ray i in

parametric form as

$$(x_1, x_2, \ldots x_K) = (P_1, P_2, \ldots P_K) + u\,(\alpha_{i1}, \alpha_{i2}, \ldots \alpha_{iK}),\ u \geq 0.$$

On the other hand, each point $(x_1, x_2, \ldots x_K)$ of ray i must also lie on all K - 1 flats of S - {i}:

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,K-1} & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{i-1,1} & a_{i-1,2} & \cdots & a_{i-1,K-1} & -1 \\ a_{i+1,1} & a_{i+1,2} & \cdots & a_{i+1,K-1} & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{K1} & a_{K2} & \cdots & a_{K,K-1} & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_K \end{bmatrix} = \begin{bmatrix} a_{1K} \\ \cdot \\ \cdot \\ \cdot \\ a_{i-1,K} \\ a_{i+1,K} \\ \cdot \\ \cdot \\ \cdot \\ a_{KK} \end{bmatrix}.$$

Combining these two equations we obtain

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,K-1} & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{i-1,1} & a_{i-1,2} & \cdots & a_{i-1,K-1} & -1 \\ a_{i+1,1} & a_{i+1,2} & \cdots & a_{i+1,K-1} & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{K1} & a_{K2} & \cdots & a_{K,K-1} & -1 \end{bmatrix} \begin{bmatrix} \alpha_{i1} \\ \alpha_{i2} \\ \cdot \\ \cdot \\ \cdot \\ \alpha_{iK} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix}.$$

To solve this system of K - 1 equations in K unknowns we take advantage of the following property of cofactors:

$$\sum_{j=1}^{K} b_{ij}\, \mathrm{cof}_{mj}(B) = \begin{cases} \det(B), & \text{if } i = m \\ 0, & \text{if } i \neq m. \end{cases} \qquad (8)$$

where $\mathrm{cof}_{ij}(B)$ is the cofactor of b_{ij}. It follows that the general solution is

$$(\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{iK}) = c_i \times (\mathrm{cof}_{i1}(B), \mathrm{cof}_{i2}(B), \ldots, \mathrm{cof}_{iK}(B)),$$

where c_i is a constant for each i. The choice of c_i is not entirely arbitrary since SGN(c_i) determines whether ray i points up or down. The correct choice for c_i satisfies

$$P_K + \alpha_{iK} < \sum_{j=1}^{K-1} a_{ij} (P_j + \alpha_{ij}) + a_{iK}$$

because each point of ray i (beyond point P) should lie below the half-space i. Using the definition of P and the above properties of cofactors we can prove that $0 \le c_i \det(B)$. The vectors α are therefore chosen to be

$$(\alpha_{i1}, \alpha_{i2}, \ldots, \alpha_{iK}) = \det(B) \times (\mathrm{cof}_{i1}(B), \mathrm{cof}_{i2}(B), \ldots, \mathrm{cof}_{iK}(B)).$$

Theorem 8: Let the UPPER half-space M, the set S of K UPPER K-dimensional half-spaces, point P, and K rays r_i be as given above. Half-space M is redundant with respect to the K half-spaces of S iff

$$P_K \le \sum_{j=1}^{K-1} a_{Mj} P_j + a_{MK} \tag{9}$$

and

$$\sum_{j=1}^{K-1} \alpha_{ij} a_{Mj} \ge \alpha_{iK}, \quad i=1, \ldots, K. \tag{10}$$

Proof: Half-space M is redundant iff flat M lies above the rays r_i, i = 1, . . ., K. We can express this condition as

$$P_K + u\alpha_{iK} \le \sum_{j=1}^{K-1} a_{Mj} (P_j + u\alpha_{ij}) + a_{MK}, \quad \forall u > 0, \; i = 1, \ldots, K.$$

If M is redundant, then we can obtain condition (9) by simply setting u = 0. Condition (10) results from dividing by u and taking the limit as $u \to \infty$. Conversely, if conditions (9) and (10) are satisfied, then M must be redundant. Simply multiply condition (10) by $u \ge 0$ and add the result to condition (9). □.

3.1.5.2. The General Transform

Before we can characterize redundancy in the transform space, we must first elucidate the important properties of the K-dimensional transform. This transform maps not only flats to points and points to flats, but also j-spaces to K-j-1-spaces.

General Transform: The transform of a j-dimensional subspace of K-space is the set of all flats that contain it. This is a K-j-1-dimensional subspace of flats. Since each flat can be readily represented as a point, however, the transform of the j-dimensional subspace is represented as a K-j-1-dimensional subspace of points.

Theorem 9: The general transform preserves incidence. In other words, if a j_1-dimensional subspace of K-space is a subspace of a j_2-dimensional subspace, then the transform of the j_2-dimensional subspace is a subspace of the transform of the j_1-dimensional subspace.

Proof: We may interpret the j_1-dimensional subspace as an intersection of K-j_1 flats and the j_2-dimensional subspace as an intersection of K-j_2 flats. That is, we may define define the j_1-dimensional subspace by

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,K-1} & -1 \\ a_{21} & a_{22} & \cdots & a_{2,K-1} & -1 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{K-j_2,1} & a_{K-j_2,2} & \cdots & a_{K-j_2,K-1} & -1 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{K-j_1,1} & a_{K-j_1,2} & \cdots & a_{K-j_1,K-1} & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_K \end{bmatrix} = - \begin{bmatrix} a_{1K} \\ a_{2K} \\ \vdots \\ a_{K-j_2,K} \\ \vdots \\ a_{K-j_1,K} \end{bmatrix}$$

and the j_2-dimensional subspace by

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,K-1} & -1 \\ a_{21} & a_{22} & \cdots & a_{2,K-1} & -1 \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{K-j2,1} & a_{K-j2,2} & \cdots & a_{K-j2,K-1} & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_K \end{bmatrix} = - \begin{bmatrix} a_{1K} \\ a_{2K} \\ \cdot \\ \cdot \\ \cdot \\ a_{K-j2,K} \end{bmatrix}.$$

The transform of the j_1-dimensional subspace is the set of points of the form

$$\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_K \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{K-j2,1} & \cdots & a_{K-j1,1} \\ a_{12} & a_{22} & \cdots & a_{K-j2,2} & \cdots & a_{K-j1,2} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ a_{1K} & a_{2K} & \cdots & a_{K-j2,K} & \cdots & a_{K-j1,K} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ u_{K-j2} \\ \cdot \\ \cdot \\ \cdot \\ u_{K-j1-1} \\ 1 - \sum_{i=1}^{K-j1-1} u_i \end{bmatrix}$$

and the transform of the j_2-dimensional subspace is the set of points of the form

$$\begin{bmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_K \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{K-j2,1} \\ a_{12} & a_{22} & \cdots & a_{K-j2,2} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{1K} & a_{2K} & \cdots & a_{K-j2,K} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ u_{K-j2-1} \\ 1 - \sum_{i=1}^{K-j1-1} u_i \end{bmatrix}.$$

It is easy to see that the transform of the j_2-dimensional subspace is a subspace of the transform of the j_1-dimensional subspace. □.

3.1.5.3. Redundancy in the Transform Space

Having presented both a characterization of redundancy of a half-space and the general duality transform, we can now interpret the conditions for redundancy in the transform space. The K flats bounding the half-spaces of S transform to K points by the formulas

$$x_K = \sum_{j=1}^{K-1} a_{ij}x_j + a_{ik} \rightarrow (a_{i1}, a_{i2}, \ldots, a_{iK}),\ i = 1, \ldots, K,$$

and the flat bounding half-space M transforms to *point* M by

$$x_K = \sum_{j=1}^{K-1} a_{Mj}x_j + a_{Mk} \rightarrow (a_{M1}, a_{M2}, \ldots, a_{MK}).$$

Since the transform preserves incidence (Theorem 9), the flat determined by the K points $(a_{i1}, a_{i2}, \ldots, a_{iK})$, $i = 1, \ldots, K$ is simply the transform of the point P determined by the K flats of S (Equation (7)). Letting the coordinates of the transform space be $z_1, z_2, \ldots, z_K$, this flat is

$$z_K = \sum_{j=1}^{K-1} (-P_j)\, z_j + P_K.$$

We can now interpret the first condition (9) for redundancy of half-space M (with respect to S) as a condition on point M and the flat determined by the transforms of the K elements of S:

$$a_{MK} \geq \sum_{j=1}^{K-1} (-P_j)\, a_{Mj} + P_K.$$

We have therefore proved

> Theorem 10: The first condition (9) for redundancy of UPPER half-space M with respect to the K UPPER half-spaces of S requires that point M (in the transform space) lie above the flat determined by the K

points $(a_{i1}, a_{i2}, \ldots, a_{iK})$, $i = 1, \ldots, K$.

The other conditions for redundancy of M (the "betweenness of slopes" conditions (10)) map to a set of K half-spaces in K-1-space

$$\sum_{j=1}^{K-1} \alpha_{ij} z_j \geq \alpha_{iK}$$

in which the point $(a_{M1}, a_{M2}, \ldots, a_{M,K-1})$ must lie. The (K-1-dimensional) points that satisfy these K conditions are

$$\begin{bmatrix} z_1 \\ z_2 \\ \cdot \\ \cdot \\ \cdot \\ z_{K-1} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{K-1,1} \\ a_{12} & a_{22} & \cdots & a_{K-1,2} \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ a_{1K} & a_{2K} & \cdots & a_{K-1,K} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ \cdot \\ \cdot \\ \cdot \\ u_{K-1} \end{bmatrix} \qquad (11)$$

where

$$\sum_{i=1}^{K} u_i = 1 \quad \text{and} \quad u_i \geq 0, \; i=1, \ldots, K. \qquad (12)$$

That is, the points that satisfy the "betweenness of slopes" conditions must lie inside the convex hull of the K-1-dimensional points $(a_{i1}, a_{i2}, \ldots, a_{i,K-1})$, $i = 1, \ldots, K$. We prove this assertion as follows. Any point in K-1-space can be written in the form of equation (11) if the restrictions of (12) are ignored. (We have, in fact, one extra degree of freedom.) The "betweenness of slopes" conditions are therefore

$$\sum_{j=1}^{K-1} \alpha_{ij} z_j = \sum_{j=1}^{K-1} \det(B)\, \mathrm{cof}_{ij}(B) \sum_{h=1}^{K} a_{hj} u_h \geq \alpha_{iK} = \det(B)\, \mathrm{cof}_{iK}(B), \quad i = 1, \ldots, K.$$

which reduce to

$$\det(B) \sum_{h=1}^{K} u_h \sum_{j=1}^{K-1} \mathrm{cof}_{ij}(B) b_{hj} \geq \det(B)\, \mathrm{cof}_{iK}(B), \quad i = 1, \ldots, K$$

But since all entries in the Kth column of B are -1, the properties of cofactors (8) allow us to reduce this further to

$$\det(B) \left[\mathrm{cof}_{iK}(B) \sum_{h=1}^{K} u_h + \det(B) u_i \right] \geq \det(B)\, \mathrm{cof}_{iK}(B), \quad i = 1, \ldots, K \qquad (13)$$

This is certainly satisfied for all I if the restrictions of (12) hold; the converse is also true. Any point Q that does not satisfy (12) can not lie in the convex hull of the points $(a_{i1}, a_{i2}, \ldots, a_{i,K-1})$, $i = 1, \ldots, K$. Here we can take advantage of the extra degree of freedom mentioned above to express the point Q as a linear combination of these K points such that

$$\sum_{h=1}^{K} u_h = 1 \qquad \text{but} \qquad u_i < 0 \text{ for some } i.$$

It is easy to see that such a point does not satisfy the conditions of (13) and is therefore not redundant with respect to S.

We have just proved

Theorem 11: The "betweenness of slopes" conditions (10) for redundancy of UPPER half-space M with respect to the K UPPER half-spaces of S require that the point $(a_{M1}, a_{M2}, \ldots, a_{M,K-1})$ lie inside the convex hull of the points $(a_{i1}, a_{i2}, \ldots, a_{i,K-1})$, $i = 1, \ldots, K$.

Combining Theorems 10 and 11 we obtain

Theorem 12: UPPER half-space M is redundant with respect to the K UPPER half-spaces of S iff point M lies directly above some point in the convex hull of the points $(a_{i1}, a_{i2}, \ldots, a_{i,K})$, $i = 1, \ldots, K$.

3.1.5.4. Redundancy Among N Half-spaces

In this section we characterize redundancy of half-space M with respect to a set of $N \geq K$ K-dimensional half-spaces. (If N is less than K then there can not be any redundant half-spaces unless some are parallel.)

Theorem 13: Let T be a set of N UPPER K-dimensional half-spaces. UPPER half-space M is redundant with respect to the half-spaces of T iff M is redundant with respect to the half-spaces of a subset S of T that contains exactly K half-spaces.

Proof: It is clear that if M is redundant with respect to a subset S of T, then it is redundant with respect to T. We shall now establish that the converse is also true. Let U be the intersection of the N UPPER half-spaces of T. U is a convex polytope because it is an intersection of half-spaces. If a half-space M is redundant then its boundary (flat M) lies completely above U. Let V be the point of U that is closest to flat M. (If the closest point is not unique, then let V be any vertex of U that is in the set of closest points.) Since U is a convex polytope, V is (or can be chosen to be) a vertex of U. Let S be the set of half-spaces whose boundaries meet at point V. If the half-spaces of T are in general position, then S contains exactly K half-spaces. Since U is convex, we can travel from vertex V along the boundary of U in any direction and the distance to flat M will be nondecreasing. It follows that the boundary of U can never intersect flat M and therefore half-space M is redundant with respect to the K half-spaces of S. □.

We have just characterized redundancy of a half-space M with respect to N half-spaces in terms of redundancy with respect to K half-spaces. But M is redundant with respect to a set of K half-spaces iff point M lies above a point in the convex hull of the points to which the K half-spaces transform. We have therefore

Theorem 14: An UPPER half-space M is redundant with respect to a set T of N UPPER K-dimensional half-spaces iff the point M (to which half-space M transforms) lies directly above a point in the convex hull of the N points of the transform of the N half-spaces of T.

Since by Theorem 9 the components of the convex hull correspond to components of the intersection of half-spaces, we have

Theorem 15: If H(N,K) is the time to construct the convex hull of N points in K-space, then N UPPER K-dimensional half-spaces can be intersected in O(H(N,K)) time.

3.1.6. Open Problems

There are still a few open problems concerning intersection of half-spaces.

1. The time to intersect N (UPPER) K-dimensional half-spaces depends on the time H(N,K) to construct the convex hull of N K-dimensional points. We know that the worst-case time complexity is $\Theta(N \log N)$ for two and three dimensions but for four or more dimensions only the $\Omega(N^2)$ time lower bound has been proven. (See Section 1.1.1 for a description of our knowledge of convex hulls.) For $K \geq 4$ dimensions we lack tight upper and lower bounds on H(N,K).

2. Under some conditions the *expected-time* for intersection of half-spaces may be less than the worst-case time. We may use fast expected-time convex hull algorithms to obtain fast expected-time intersection of half-space algorithms. For example, in two and three dimensions if the expected number of nonredundant half-spaces is $O(N^p)$ for some $p < 1$, then N half-spaces can be intersected in O(N) expected-time [16]. Since the convex hull of N K-dimensional points can be constructed in O(N) expected-time, if the K coordinates have independent distributions [10, 30] , we can intersect N K-dimensional half-spaces in O(N) expected-time if the N half-spaces transform to N points whose K coordinates are distributed independently. Under what other conditions may we intersect half-spaces in fast expected-time?

3. We can intersect two convex (three-dimensional) polyhedra in O(N log N) time by simply treating it as a problem of intersecting half-spaces. Can we improve this to O(N) time? (In two dimensions, convex N-gons can be intersected in O(N) time whereas it requires O(N log N) time to intersect N half-planes in the worst case.)

4. How fast can we intersect N half-spaces *on-line* in two or more dimensions? (Shamos [91] has presented an O(N log N) time planar on-line convex hull algorithm and Preparata [78] has refined that to an O(N log N) time *real-time* algorithm. Both of these algorithms update the convex hull as each point is read -- rather than operating on all N points collectively -- but the on-line algorithm may require up to O(N) time for any particular update whereas the real-time algorithm always requires at most O(log N) update time.)

3.2. Union and Intersection of Disks

We present the problem of constructing the union or intersection of a set of disks (the interiors of a set of circles) not for its applications, but because the solution presents several techniques that are useful for solving many other problems. We introduce two new transforms, inversion and embedding in a higher dimension. Inversion is used to convert problems that involve circles or spheres to problems that involve lines or planes. Embedding in a higher dimension adds another degree of freedom to the problem, which can permit application of techniques not applicable to the original problem. The disk algorithm combines inversion and embedding to transform the problem of constructing the union or intersection of a set of N disks to the problem of intersecting N three-dimensional half-spaces, which we have solved in O(N log N) time.[10] Another important feature of the algorithm is that it demonstrates an example of how a nonconvex object (the union of disks) can be represented by a convex object (an intersection of half-spaces).

3.2.1. Representation of Disks and Their Union or Intersection

A disk is the set of all points within a given positive radius R from a planar point P. If the coordinates of P are (P_x,P_y), then the disk may be represented as a triple (P_x,P_y,R) and a set of disks as a set of such triples. The best representation for the union or the intersection of a set of disks depends on what kinds of information about the union or intersection we want to retrieve efficiently. For several applications the best representation will be as an intersection of half-spaces in the transform space (to be described), but for other applications we may need a representation in the original space. We will not describe either representation in detail because we have already described the representation for intersection of half-spaces and the representation in the original space is only a slight modification

[10] Shamos and Hoey [94] describe an O(N log N) time intersection of N disks that is a modification to their algorithm for intersecting half-planes. Their algorithm unfortunately does not extend to an O(N log N) time algorithm for the union of N disks.

of the representation for polygons.

3.2.2. Lower Bound for the Union or Intersection of Disks

We prove that an algorithm that constructs the union or intersection of N disks can be used to sort. Since there is an $\Omega(N \log N)$ time lower bound for sorting (under the same model of computation used in the construction below), we have an $\Omega(N \log N)$ time lower bound for constructing the union or intersection of N disks.

Theorem 16: The construction of the union or the intersection of N disks takes $\Omega(N \log N)$ time.

Proof: Let S be a set of N reals $t_i \in [0,1)$, $i = 1, \ldots, N$. For each real t_i we have a disk d_i centered at (x_i,y_i) with radius two where

$$(x_i, y_i) = (\cos(2\pi t_i), \sin(2\pi t_i)).$$

As illustrated in Figure 3-8, the points (x_i,y_i) are all on the unit circle and the boundaries of the disks d_i are tangent to the unit circle. The union of the disks d_i is represented by a closed chain of circular arcs (ab-bc-cd-da in Figure 3-8). The order of the arcs in this chain forms a sort for the N reals t_i. Similarly, an intersection of the same N disks is represented by a closed chain of circular arcs (he-ef-fg-gh in Figure 3-8) whose order sorts the reals t_i.

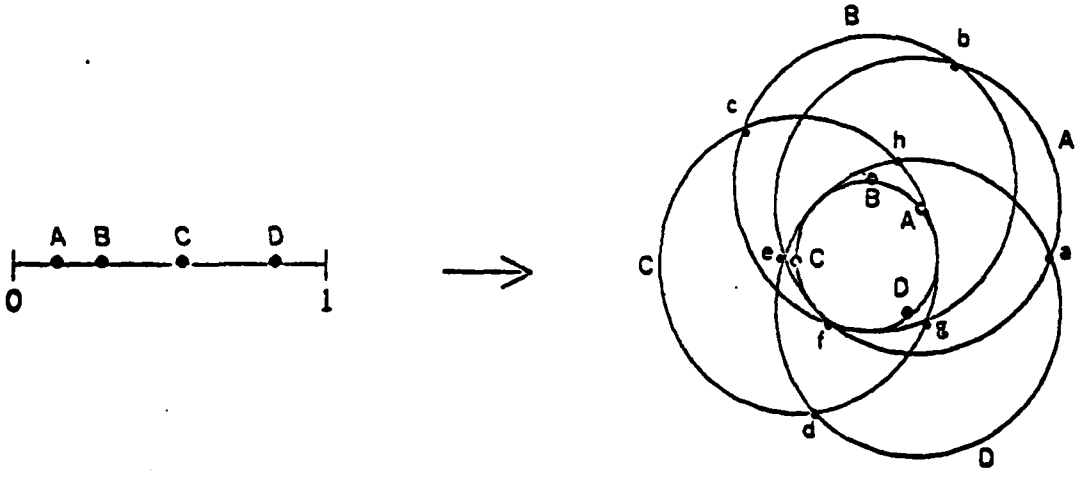


Figure 3-8: Sorting with a union or intersection of disks algorithm.

The proof of the lower bound requires an $\Omega(N \log N)$ time lower bound for sorting under the same model of computation used to construct the N disks d_i from the N reals t_i. Since the construction of circles uses the functions sine and cosine, a model of computation with only linear functions of the input is not sufficient. As for the intersection of half-spaces, we may use Friedman's [43] result that sorting has an $\Omega(N \log N)$ time lower bound even when arbitrary functions are allowed at internal nodes of the decision tree and the output functions are analytic. Since sine and cosine are analytic, the $\Omega(N \log N)$ lower bound for sorting carries over to the union and intersection of disks.

3.2.3. The Inversion Transform

Our algorithm for constructing the intersection or union of N disks is based on the properties of the inversion transform. We will first describe the two-dimensional transform and then generalize to three (and higher) dimensions. For more information on inversion we refer the reader to Dodge [36].

The inversion transform is determined by two parameters: (1) the center of inversion, and (2) the radius of inversion. For simplicity of exposition we shall assume (for now) that the center of inversion is the origin and that the radius of inversion is one. If a point P has polar coordinates (R,θ), then the inversion transform of P is

$$(R, \theta) \rightarrow (1/R, \theta).$$

Inversion maps a vector in the direction θ to another vector in the same direction but with its magnitude "inverted." Note that inversion is involutory -- application of inversion twice yields the original point. Figure 3-9 illustrates another important property of inversion in the plane. A circle that passes through the center of inversion transforms to a line that *does not* pass through the center of inversion, and vice versa. Furthermore, the interior of the circle transforms to one of the half-planes determined by that line and the exterior of the circle transforms to the other half-plane. The properties of inversion in three dimensions are analogous. The transform can be expressed in spherical coordinates as

$$(R, \theta, \phi) \rightarrow (1/R, \theta, \phi).$$

This transform is involutory, as in two dimensions, and it also transforms any *sphere* that passes through the center of inversion to a *plane* that does not pass through the center of inversion. The interior of the sphere transforms to a half-space bounded by that plane and the exterior of the sphere transforms to the other half-space.

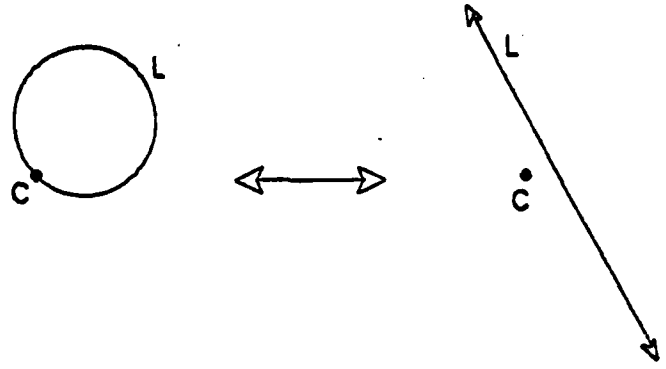


Figure 3-9: Inversion transforms lines to circles and vice versa.

3.2.4. Algorithm for Intersection or Union of Disks

We will now exploit the properties of the inversion transform to construct a fast algorithm for the union or intersection of disks. We will first describe the simple case where the circles bounding the N disks share a common point P, and then generalize the result to an arbitrary set of N disks in the plane.

Figure 3-10 illustrates the special case where the N boundary circles share a common point P. Let C_i denote disk i, for i=1,...N. Since circle i passes through point P, inversion about P transforms C_i to a half-plane H_i. It follows that the union of the N disks transforms to the union of N half-planes:

$$\bigcup_{i=1}^{N} C_i \rightarrow \bigcup_{i=1}^{N} H_i.$$

Similarly, if $\bar{X}$ denotes the complement of X, then the union of the C_i transforms by

$$\bigcup_{i=1}^{N} C_i \rightarrow \bigcup_{i=1}^{N} H_i = \overline{\bigcap_{i=1}^{N} \overline{C}_i}$$

which can also be solved as an intersection of half-planes. Since N half-planes can be intersected in O(N log N) time, we have

> Theorem 17: The union or intersection of N disks can be constructed in O(N log N) time when the circles bounding the N disks share a common point P.

In general, however, the N circles will not have any point P in common. In fact, the disks may be entirely disjoint. Nevertheless, we can manipulate the general problem so that it looks sufficiently like the special case that inversion can be applied to obtain a fast algorithm.

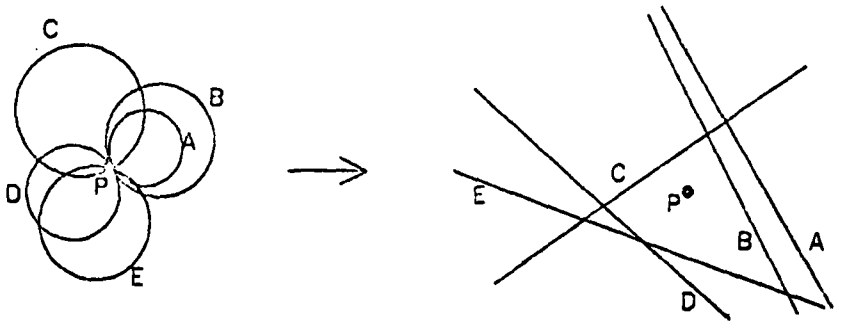


Figure 3-10: Special case for inversion: All boundary circles meet at point P.

> Theorem 18: The union or intersection of N disks can be represented as a convex polyhedron in O(N log N) time.

> Proof: We illustrate the construction in Figure 3-11. We first embed the N disks in three dimensions with the disks all located in the xy plane. We then choose an arbitrary point P that does not lie in the xy plane. For each disk c there is a unique sphere that passes through point P and that intersects the xy plane at circle c.[11] We can thus represent the N disks

[11] A sphere is determined by four parameters, the three coordinates for its center and the radius. A circle and a noncoplanar point determine a unique sphere because requiring the sphere to pass through the circle costs three degrees of freedom and requiring the sphere to pass through the point determines the fourth.

in the xy plane by N balls whose (spherical) boundaries share a common point P. Inversion about point P transforms the N spheres to N planes, the balls to half-spaces, and and the exteriors of the balls to complementary half-spaces. The intersection of N disks is thus represented by the intersection of N half-spaces. Similarly, we represent the union of N disks by (the complement of) the intersection of N (complementary) half-spaces. Since we can intersect N half-spaces in O(N log N) time, we can represent the union or intersection of N arbitrary planar disks by a convex polyhedron in O(N log N) time. □.

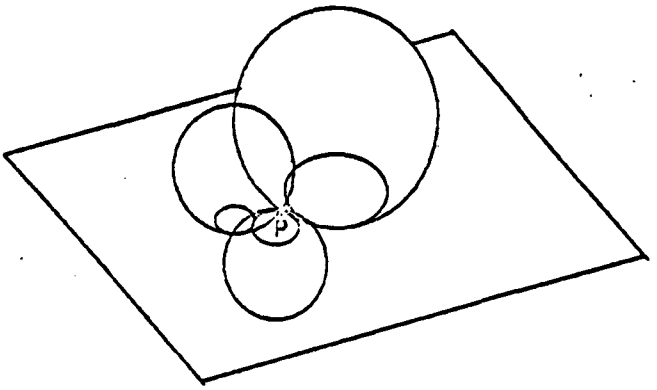


Figure 3-11: General case for intersection or union of N disks.

3.2.5. Related problems

We will now briefly describe a few issues related to the union of disks that we have not yet covered.

1. We can generalize our technique for representing the union or intersection of disks to also allow *subtracting* circular regions from a union or intersection of disks. Since each circle maps to a plane in three-space, there are two three-dimensional half-spaces that we may associate with a given circle. One half-space corresponds to the interior of the circle and the other corresponds to the exterior. We may thus represent any intersection of the interiors of circles or their complements by an intersection of half-spaces.

2. Suppose that we wanted to preprocess N disks so that given an arbitrary planar point we could quickly determine if the point lies in any

of the N disks. If we represent the union of the disks as an intersection of half-spaces, then this problem becomes the problem of determining if a point in three-space lies within a convex polyhedron. This problem is equivalent to locating a point within two planar straight-line graphs.[12] As we mentioned in the introduction, location of a point within a planar graph of size N can be done in O(log N) time given O(N log N) preprocessing time and O(N) storage [70]. We must also note that location of a point in a set of circles is a decomposable searching problem [5].

3.3. Derivation of the Point / Flat Duality

The intersection of half-spaces involves a point / flat duality and convex hulls whereas the union of disks depends on inversion (and embedding in a higher dimension). In this section we show that these techniques are closely related by deriving the point / flat duality as a limiting case of inversion, linear transforms, and a circle / point duality. In the construction we demonstrate the relationship between convex hulls and the union of disks or half-spaces.

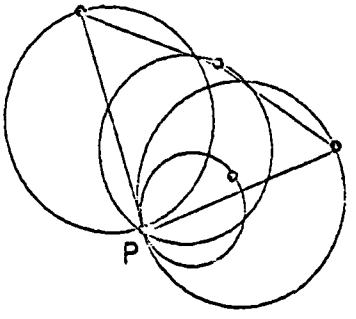


Figure 3-12: The union of (the interiors of) circles that meet at a point P.

In the previous section we constructed the union of (the interiors of) N circles

[12] Simply break the convex polyhedron into UPPER and LOWER parts and project both parts orthographically to planar graphs in the *xy* plane. If a three-dimensional point P projects to a planar point in region R of the UPPER graph and region S of the LOWER graph, then P lies within the convex polyhedron iff P lies below face R and above face S of the polyhedron.

with a point P in common by first using inversion to transform the interiors of the circles to half-planes. In Figure 3-12 we demonstrate an alternate solution:

Construction of the Union of N Disks whose Boundaries Share a Point P

1. Let C_i, i = 1, . . ., N be a set of disks whose boundary circles meet at point P. Transform each of the N disks C_i to the points P_i diametrically opposite point P. (This is the Circle / Point duality.)

2. Construct the convex hull of the set of the points P_i, i = 1, . . ., N augmented with point P.

3. A disk C_i is nonredundant in the union of the N disks iff its transform P_i is a vertex of the convex hull. To obtain the circular arcs defining the union we simply transform the vertices of the convex hull back to disks, obtaining the endpoints of the arcs from the points where neighboring disks intersect.

Since the most expensive step in the algorithm is the construction of the convex hull, the total time is O(N log N). We will omit the proof that the algorithm correctly constructs the union of the N circles because, as we shall see, it is essentially equivalent to our proof of the algorithm for intersection of half-spaces.

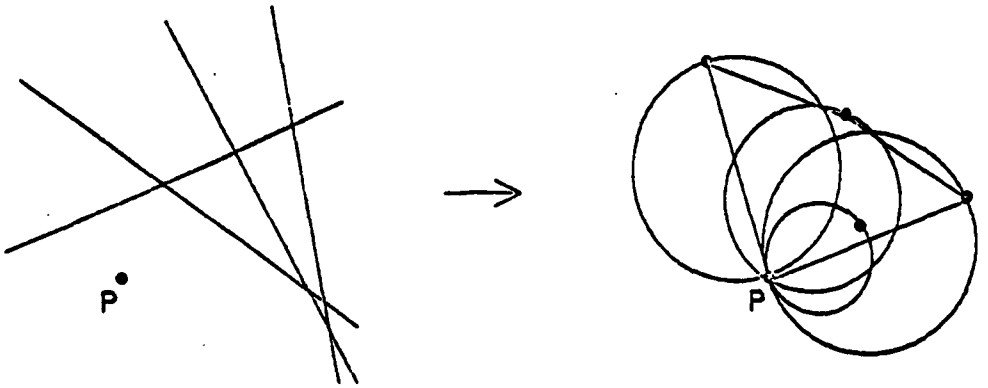


Figure 3-13: The union of half-planes transforms to the union of disks.

In Figure 3-13 we illustrate the union of a set of half-planes, each of which *does*

not contain point P.[13] By inverting the half-planes about point P we transform the union of half-planes problem to a union of disks problem where the circles bounding the disks all meet at point P. But as we just saw, we can construct the union of the disks by transforming them to points and constructing the convex hull of those points (and point P). We will now describe the transform algebraically.

To make the mathematics simpler we can assume that the point P lies on the negative y-axis:

$$P = (0,R), \quad R < 0.$$

Our algorithm inverts each half-plane about point P, obtaining the interior of a circle, and then maps the circle to the point diametrically opposite P. We express this as

$$y = a_i x + b_i \quad \rightarrow \quad (-a_i/(b_i-R), R + 1/(b_i-R)), \; i = 1, \ldots, N.$$

To show how this relates to the point / flat duality that we used in Section 3.1 for intersecting half-planes, we must now use a trick.

Recall from Step 3 of our algorithm for constructing the union of the disks, a disk C_i is nonredundant iff its transform P_i is a vertex of the convex hull. If the convex hull is translated, stretched, or rotated the vertices will still remain vertices and the points inside the hull will remain inside the hull. In particular, the transform

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} R & 0 \\ 0 & -R^2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ R^3-R \end{bmatrix}$$

applied to the point P and points P_i, $i = 1, \ldots, N$ will not affect our determination of which points lie on the convex hull and which do not. Our new transform, inversion of a half-plane about point P followed by the circle / point duality and our linear transform, can be expressed as

---

[13] We can alternately think of the union of half-planes as the complement of the intersection of the complementary half-planes.

$$y = a_i x + b_i \rightarrow \left(\frac{-a_i R}{b_i - R}, \frac{-b_i R}{b_i - R}\right).$$

Note also that point P transforms by

$$P = (0,R) \rightarrow (0, -R).$$

Taking the limit as $R \rightarrow -\infty$ we obtain

$$y = a_i x + b_i \rightarrow (a_i, b_i)$$

and for point P

$$P = (0,R) \rightarrow (0, +\infty).$$

The vertices of the convex hull of the points (a_i,b_i), $i = 1, \ldots, N$ augmented with $(0,+\infty)$ are the vertices on the *bottom part* of the convex hull of the points (a_i,b_i) (and $(0,+\infty)$). This is exactly the characterization given for intersection (or union) of half-planes in Section 3.1. We have

Theorem 19: The point / flat duality is a limiting case of inversion followed by the circle / point duality and a linear transform.

3.4. Summary

In this chapter we have presented several important transforms and techniques:

- A point / flat duality --

  This maps points to flats and flats to points. Since there are already a number of algorithms for point problems, this transform finds greatest use in transforming problems that are expressed (or expressible) in terms of flats to problems that involve points. Two of the important properties of the duality that we described are

  (a) it preserves above/belowness between points and flats, and

  (b) it preserves distance between points and flats in the x_K coordinate, and thus preserves incidence.

- Embedding into a higher dimension --

This gives another degree of freedom to the problem that allows application of techniques not applicable to the original problem -- circles can become spheres, lines can become planes, etc. Since the higher-dimensional object has a degree of freedom that we can choose arbitrarily, the objects can be chosen to conform to an expecially simple case (all N spheres have a point in common). In most of the applications of embedding in a higher dimension (in this thesis) the problem is first expressed in terms of circles and then embedded into a higher dimension and re-expressed in terms of spheres.

- Inversion --

Inversion is a circular transform; circles map to circles (where a line is considered to be a circle of infinite radius). In particular,

(a) a circle that passes through the center of inversion maps to a line that does not pass through the center of inversion, and

(b) the interior of a circle that contains the center of inversion maps to the exterior of a circle that contains the center of inversion.

In three dimensions we have the same relationships between spheres and planes and in K dimensions between K-spheres and flats. Inversion is also involutory -- application of it twice yields the original object. The main use for inversion (in this thesis) is transforming problems that are expressed (or expressible) in terms of circles or spheres to problems that involve lines or planes and for which fast algorithms are known.

In this chapter we have also seen an application of convex hulls in the intersection of half-spaces and in the relationship between inversion, linear transforms, and the point / flat duality. In later chapters convex hulls will be applied to several other problems that involve a "network" of linear parts.

4. Construction of Nearest and Farthest Point Diagrams

Geometric transforms are important tools in the construction of many different kinds of nearest and farthest point diagrams. These diagrams include (nearest and farthest point) Voronoi diagrams in Euclidean and spherical spaces, and the (nearest and farthest) edge diagrams of a convex polygon. Each of these diagrams is a tesselation of space into sets of points closest to (or farthest from) the elements (points or edges) defining the diagram. We will find in each case that it is useful first to express the problem in terms of a set of circles that define the diagram and then to embed the problem into a higher-dimensional Euclidean space, which allows us to use techniques not applicable to the original problem. In the following sections we describe these diagrams and algorithms for constructing them.

4.1. Euclidean Voronoi and Delaunay Diagrams

Voronoi diagrams (also called Thiessen diagrams or Dirichlet tesselations) find application in cluster analysis [51], construction of contour maps [29], construction of Euclidean minimal spanning trees [93], crystal growth [46], and several interesting problems in geometry [91]. We can easily show an $\Omega(N \log N)$ time worst-case lower bound by demonstrating that any algorithm that constructs a Voronoi diagram can be used to sort [91]; the challenge is to construct an $O(N \log N)$ time algorithm. Shamos [89] describes an $O(N \log N)$ time divide-and-conquer algorithm for construction of the planar Euclidean Voronoi diagram and Lee and Wong [67] describe an $O(N \log N)$ time algorithm for the L_1 and L_∞ metrics in the plane. Drysdale and Lee [37] present an $O(N c^{(\log N)^{1/2}})$ time algorithm for the Voronoi diagram of line segments (and other planar objects), which they have improved to $O(N \log^2 N)$ time. Kirkpatrick [59] presents an $O(N \log N)$ time algorithm for constructing the Voronoi diagram of N planar line segments. Shamos [89], Lee and Preparata [66], Preparata [80], and Lipton and Tarjan [70] have produced fast algorithms for searching a Voronoi diagram (or any other straight-line planar graph).

The algorithm for construction of a Euclidean Voronoi diagram that we describe

below [23] is not only a very useful result in itself, but it also serves as an example of the use of several important algorithmic tools. We use the technique of embedding into a higher dimension and applying inversion (as in the algorithm for union of circles) and we also use a convex hull algorithm (as in the intersection of half-spaces).

4.1.1. Definition of Planar Voronoi and Delaunay Diagrams

Let S be a set of N planar points such that no four points are cocircular.[14] A *nearest point* planar Voronoi diagram of S, as pictured in Figure 4-1, is a polygonal network of N regions. For each point P_i of S, region R_i is the set of all points of the plane that are closer to point P_i than to any of the other N-1 points of S. Given an arbitrary point P in the plane, we can thus determine which of the N points of S is closest to P by determining which of the N regions contains point P. The vertices of these polygonal regions are called *Voronoi points* and the polygonal boundaries of the regions are called *Voronoi polygons*. If a Voronoi polygon is bounded, then it is constructed entirely from edges of the Voronoi diagram. If it is unbounded, then it includes two rays of the diagram.

Each Voronoi point V of the nearest point diagram is equidistant from the three points of S that are nearest V. This yields a property of Voronoi diagrams that we exploit in the algorithm of Section 4.1.3.

> A point V is a Voronoi point (of the nearest point Voronoi diagram of S) iff it is the center of a circle that passes through three points of S but does not contain any of the other N - 3 points of S.

The edges of a Voronoi diagram connect pairs of Voronoi points whose corresponding circles meet at two common points of S. The rays are determined similarly by one Voronoi point from the nearest point diagram and one from the farthest-point Voronoi diagram (described below).

---

[14] Since any three noncollinear planar points determine a unique circle, four planar points are cocircular only in degenerate cases.

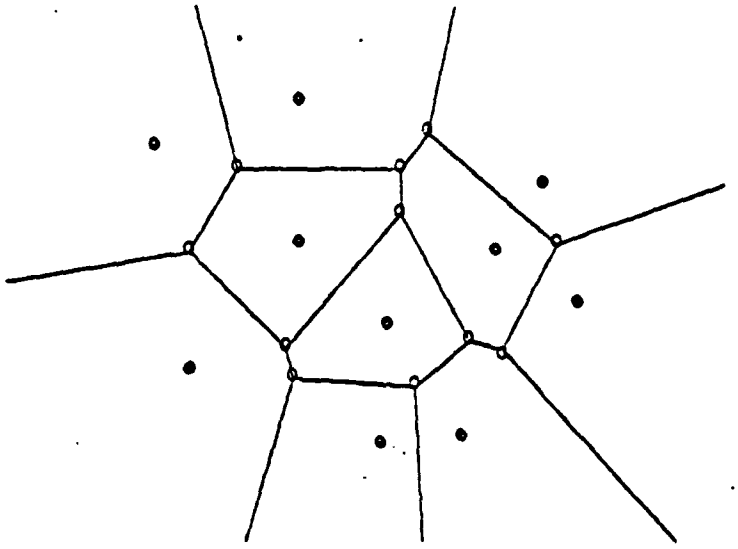


Figure 4-1: Planar Nearest Point Voronoi Diagram

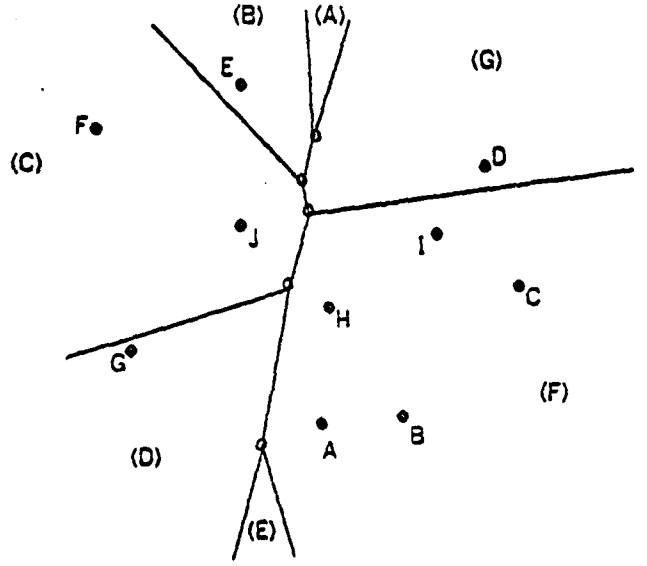


Figure 4-2: Farthest Point Voronoi Diagram

A *farthest point* planar Voronoi diagram (Figure 4-2) is also a network of polygonal

regions, but region R_i is the set of all points in the plane that are *farther* from point P_i than any other point of S. As for the nearest point diagram, there is a set of circles that define the Voronoi points for a farthest point diagram.

> A point V is a Voronoi point of the farthest point Voronoi diagram iff it is the center of a circle that passes through three of the points of S and contains *all* of the other N - 3 points.

It is important to note that only points that are vertices of the convex hull of the N points of S have nonempty farthest point regions. This is because each Voronoi point V of the farthest point diagram must be equidistant from the three points of S that are *farthest* from V. It is not possible to construct such a Voronoi point from points of S that are not on the convex hull.

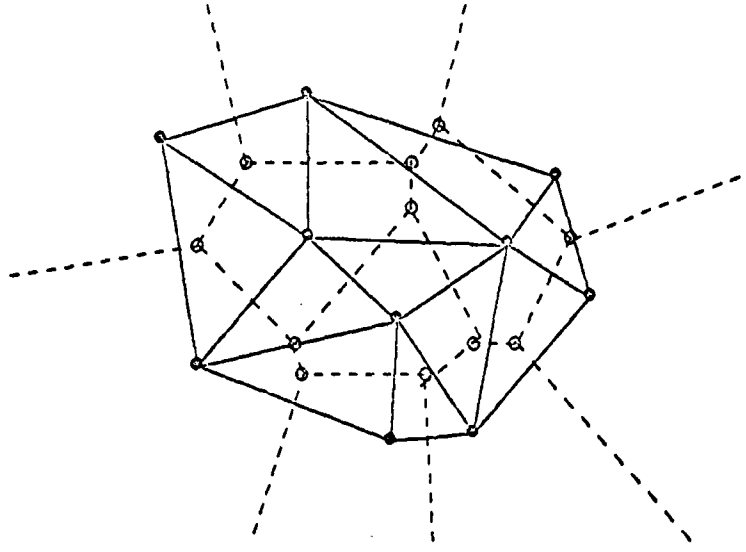


Figure 4-3: Planar Delaunay Diagram

Both the nearest and farthest point Voronoi diagrams have planar straight-line duals, called Delaunay diagrams. Figure 4-3 illustrates the dual of a nearest point Voronoi diagram and Figure 4-4 illustrates the dual of a farthest point Voronoi diagram. Both of these dual diagrams form a triangulation of the points of S. The vertices of each triangle of the (nearest point) Delaunay diagram (Figure 4-3)

determine a circle that does not contain any of the other N - 3 points of S. Similarly, the vertices of each triangle of the dual of the farthest point Voronoi diagram (Figure 4-4) determine a circle that contains all of the other N - 3 points of S.

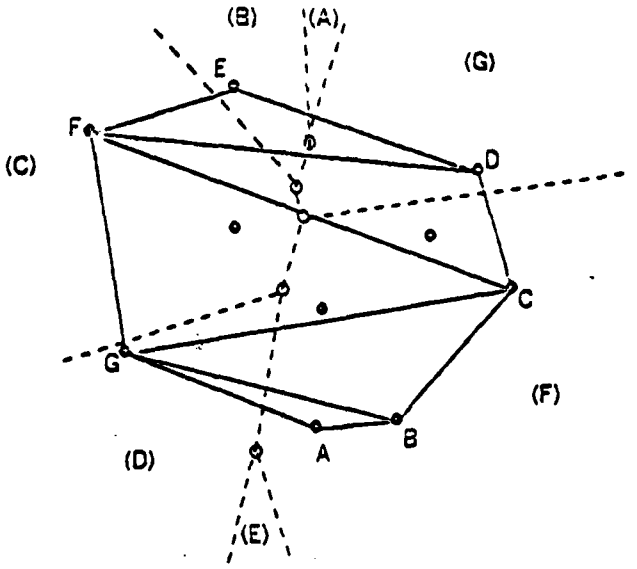


Figure 4-4: Dual of a Farthest Point Voronoi Diagram

Since the nearest and farthest point Voronoi diagrams and their duals are planar graphs, the number of Voronoi points is at most 2N - 4 and the number of edges is at most 3N - 6 for N > 2 [50]. Shamos [89, 91] and Shamos and Hoey [93] give more information on Voronoi and Delaunay diagrams.

4.1.2. Representation of Voronoi and Delaunay Diagrams

The representation of Voronoi and Delaunay diagrams should enable us to access conveniently all of the proximity information stored in the diagrams. This does not require an exotic data structure -- we can accomplish it with a well-chosen set of arrays. Since the rays of both the nearest and farthest point Voronoi diagrams are determined by one nearest Voronoi point and one farthest Voronoi point, it is easiest for us to describe a representation for both the nearest and farthest point diagrams

simultaneously. One such representation is the five-tuple

(S, V, E, StoE, EPtr),

where

S[I,1] is the X coordinate and S[I,2] is the Y coordinate of point P_I of S,

V[I,1] is the X coordinate, V[I,2] is the Y coordinate of the Ith Voronoi point and V[I,3] is a one bit flag that distinguishes the Voronoi points of the nearest point diagram from those of the farthest point diagram,

E[I,1] and E[I,2] point to the two Voronoi points of V that determine edge I. Note that for a ray one point will be from the nearest point diagram and one will be from the farthest point diagram. Furthermore, we store the edges sorted counterclockwise to make it convenient to determine the Voronoi polygons associated with each point J of S. The first few edges of E define the edges of the nearest (and farthest) Voronoi polygons of point 1 of S, the succeeding edges of E define the Voronoi polygon(s) for point 2 of S, and so on.

EPtr[I,1] points to the first edge in array E of the nearest point Voronoi polygon for point I of S. Similarly, EPtr[I,2] points to the first edge in array E of the farthest point Voronoi polygon for point I of S (if it exists).

VtoS[I,1], VtoS[I,2], and VtoS[I,3] point to the three points of S that determine Voronoi point I.

The representation for the dual nearest and farthest point diagrams is equivalent to the representation for the nearest and farthest point Voronoi diagrams.

4.1.3. Planar Voronoi Diagram Algorithm

We here combine the tools of the previous sections to produce an O(N log N) time algorithm for constructing a Voronoi diagram of a set S of N planar points. The algorithm takes advantage of the fact that the Voronoi points of the nearest point diagram can be represented by a set of circles that each (i) pass through three of the N points of S, and (ii) do not contain any of these N points in the interior. As suggested in Section 3.4, when we have a problem that is expressed in terms of circles, it may be profitable to try to apply inversion to obtain an equivalent problem

that involves linear components rather than circular components. In this case (as for the union of disks in Section 3.2) we must first embed the planar problem into three dimensions and express the new problem in terms of spheres before we can profitably apply inversion. This is because inversion transforms circular (or spherical) components into linear components only if the circle (or sphere) passes through the center of inversion. We need the extra degree of freedom that we obtain by embedding in a higher dimension to satisfy this condition. The linear (component) problem that we obtain turns out to be simple: construct the convex hull of N transformed points. Furthermore, we can also obtain the farthest point diagram from the same convex hull.

Algorithm for Construction of a Planar Voronoi Diagram

1. Let S be a set of N planar points located in the xy plane of three-space. Pick a point P in three-space that is not in the xy plane [15].

2. Choose any radius of inversion R > 0 and then invert the N points of S with respect to point P and radius R. (Section 3.2.3 describes inversion.) Call this new set of N points S'.

3. Construct the convex hull of the points in S' in O(N log N) time (by the algorithm of Preparata and Hong [83]). All N of the points of S' will be on the convex hull because inversion about P maps all points of the xy plane to a sphere with P at the apex. (See Figure 4-5.) Let f be the number of faces on the convex hull and let the edge (if any) joining faces F_i and F_j be denoted E_{ij}.

4. Each of the f faces F_i of the convex hull determines a plane in three-space. Invert these f planes (with respect to center of

---

[15] Although, mathematically, any point P outside the xy plane will work, we may encounter excessive round-off error in a computer if P is badly chosen. We want to choose x and y coordinates that (approximately) center P over the convex hull of the N points and choose the z coordinate so that it is not too close to the xy plane (which clusters the transformed points around P) and also not too far away from the xy plane (which makes all of the transformed points approximately coplanar). If x_{max}, x_{min}, y_{max}, and y_{min} are the max and min x and y coordinates among all of the N planar points then let $P_x = (x_{max} + x_{min})/2$, $P_y = (y_{max} + y_{min})/2$, and $P_z = \max((x_{max}-x_{min}), (y_{max}-y_{min})) / 4$. Since we can find easily x_{max}, x_{min}, x_{max} and x_{min} in O(N) time we can choose a good point P in O(N) time.

inversion P and radius R) to obtain f spheres that intersect the xy plane in f circles. The centers of these circles are the Voronoi points V_j. To distinguish nearest and farthest Voronoi points we perform the following simple test:

> The plane of face F_i determines two half-spaces, one that contains the entire convex hull and one that contains none of it. Let half-space H_i be the one that contains the convex hull. If H_i contains point P then V_i is a Voronoi point of the nearest point Voronoi diagram. Otherwise, V_i is a Voronoi point of the farthest point diagram.

5. We obtain the Voronoi points V_i from the faces F_i of the convex hull, but to construct the remainder of the Voronoi diagram we must examine the edges. Each edge E_{ij} of the convex hull corresponds to a segment of the nearest point diagram, a segment of the farthest point diagram, or a ray (for both diagrams). To determine for an edge E_{ij} which of these three possibilities is true we use the following rules:

   - If V_i and V_j are both nearest Voronoi points, then there is a line segment connecting V_i and V_j in the nearest point diagram.

   - If V_i and V_j are both farthest Voronoi points, then there is a line segment connecting V_i and V_j in the farthest point diagram.

   - If V_i is a closest Voronoi point and V_j is a farthest Voronoi point, then V_i and V_j determine a ray in both the nearest and farthest point Voronoi diagrams. The points V_i and V_j determine a line, and the desired ray for the nearest point Voronoi diagram is the part of that line that starts at point V_i and does <u>not</u> include point V_j. The ray starting at point V_j that does not include point V_i is for the farthest point Voronoi diagram.

Although it is clear that the above algorithm requires only O(N log N) time and O(N) storage, it is not immediately obvious that it actually constructs the nearest (or farthest) point Voronoi diagram. We must explain (1) why the centers of the circles (generated in Step 4 above) are the Voronoi points and (2) why the connection rules (Step 5) for Voronoi points work. We will now show this for the case of the nearest point diagram. The argument for the farthest point diagram is similar.

<u>Theorem 20:</u> The centers of the circles generated in Step 4 of the above algorithm are the Voronoi points.

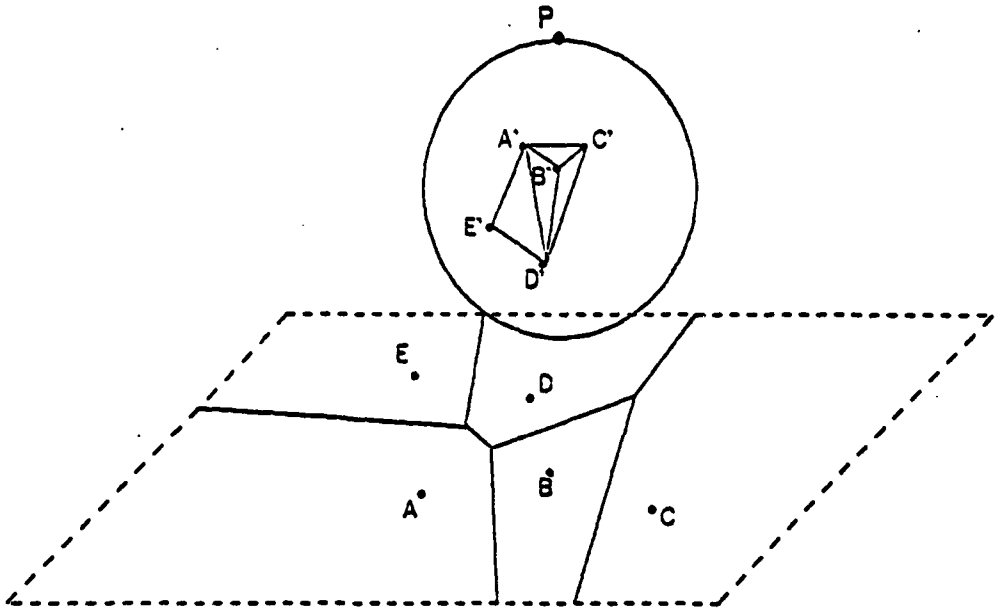


Figure 4-5: Planar Voronoi diagram and corresponding convex hull.

Proof: The proof is in two parts: (1) all of the points generated in Step 4 are Voronoi points, and (2) all of the Voronoi points are generated in Step 4. To prove that the circles generated in Step 4 are centered at the Voronoi points we must show that (a) these circles each pass through three of the N points of S and (b) do not contain any of the other N - 3 points in the interior (Section 4.1.1). Part (a) follows from the fact that inversion is involutory (Section 3.2.3). We prove part (b) by contradiction. Assume that the circle passing through points A, B, and C of S contains another point $Q \in S$ in its interior. This places point Q inside the sphere determined by points A, B, C, and P. When we invert about point P, the point Q' is separated from point P by the plane determined by points A', B', and C'. Since the plane A'B'C' does not determine a half-space that contains point P and also contains all of the other N - 3 points of S' it cannot be a face of the convex hull that determines a Voronoi point of the nearest point Voronoi diagram.

To prove that Step 4 generates all of the Voronoi points we simply use the reverse argument. If V_i is a Voronoi point then the circle for V_i transforms (by inversion) to a face of the convex hull of S'. Let A, B, and C be the points of S that determine Voronoi point V. The points A', B', and C' of S' determine a plane that contains all of the points of S' because all

N - 3 other points of S lie outside the circle determined by points A, B, and C. □

Theorem 21: Step 5 of the algorithm correctly obtains the edges of the Voronoi diagram.

Proof: The proof is in two parts: (1) all of the edges generated by Step 5 are edges of the diagram and (2) all edges of the diagram are generated by Step 5. An edge E_{ij} of the convex hull that separates (nearest point) faces F_i and F_j maps to a line segment between Voronoi points V_i and V_j. But the circles corresponding to Voronoi points V_i and V_j meet at two of the N points of S because the corresponding faces F_i and F_j share an edge E_{ij}. This is exactly the characterization given for edges of the Voronoi diagram in Section 4.1.1. Similarly, the rays are determined by edges E_{ij} where V_i is a nearest Voronoi point and V_j is a farthest Voronoi point (or vice-versa). In this case, too, the circles corresponding to V_i and V_j meet at two of the N points of S. □

Since the Voronoi points and edges (and rays) connecting the Voronoi points are correctly generated by the above algorithm, we have just proven

Theorem 22: The algorithm constructs the Voronoi diagram in O(N log N) time.

4.1.4. Fast Expected-Time Algorithms

The most expensive part of the algorithm for construction of a Voronoi diagram is the construction of the convex hull. If the convex hull can be constructed in fast *expected* time, then the Voronoi diagram can be constructed in fast expected-time. The O(N) expected-time algorithms of Bentley and Shamos [16], Eddy [39], or Floyd [40] do not apply because their results depend on a sublinear expected number of points on the convex hull, and for the Voronoi diagram algorithm there are always N vertices on the convex hull.

Bentley, Weide, and Yao [18], on the other hand, describe how a planar Voronoi diagram can be constructed in linear expected-time. The only condition is that the probability density of the underlying distribution must be bounded above and below by (nonzero) constants. The algorithm does not make use of inversion. Instead, it

applies an extension of Weide's [99] technique for an O(N) expected-time sort to the planar Voronoi diagram problem.

4.1.5. Higher Dimensions

The K-dimensional Voronoi diagram algorithm is an extension of the planar algorithm. We first embed the N K-dimensional points of S in K+1-space and then invert them to N K+1-dimensional points S'. We then construct the convex hull of S' and obtain the Voronoi diagram by transforming the parts of the convex hull back to K-space. To transform back to K-space we first invert each hyperface of the convex hull to obtain a set of K+1-spheres whose intersection with K-space is a set of K-spheres. These K-spheres each pass through K+1 points of S and are centered at the Voronoi points. We obtain the other components of the K-dimensional Voronoi diagram by connection rules similar to those in Step 5 of the algorithm in Section 4.1.3. For example, if the K-sphere for Voronoi point V_i passes through K of the K+1 points determining the K-sphere for Voronoi point V_j, then we draw a one-dimensional edge between V_i and V_j. If the spheres for a set of three or more Voronoi points share K-1 points of S, then we draw a two-dimensional edge between the Voronoi points of that set. (A two-dimensional edge between L points is a convex polygon with L vertices.) The rules for three and higher dimensional edges are similar. The time complexity of the K-dimensional Voronoi diagram algorithm is dominated by the time to construct a K+1-dimensional convex hull of N points. (See Section 1.1.1 for references to several convex hull algorithms.)

4.2. Spherical Nearest and Farthest Point Voronoi Diagrams

Voronoi diagrams are useful for solving several closest or farthest point geographic problems. If, however, the area covered by the points is large, then we must take the curvature of the earth into consideration. The most obvious approximation to use for the earth is a sphere. Nearest and farthest point Voronoi diagrams on a sphere are defined in a manner analogous to their planar counterparts and the algorithms for constructing them provide an interesting comparison with

those for the planar case. For example, we obtain the farthest point Voronoi diagram of a set of N spherical points S by simply applying a nearest point algorithm to a set S' of N points *diametrically opposite* the points of S. There are two different methods for constructing these diagrams in O(N log N) time. One involves an intersection of half-spaces and the other obtains the dual of the Voronoi diagram from the convex hull of the spherical points. We will describe only the second algorithm because it is simpler.

We take advantage of the fact that the spherical Voronoi diagram, as well as the planar Voronoi diagram, can be expressed in terms of a set of circles: the Voronoi points are the centers of the circles (on the sphere) that (i) pass through three of the N spherical points, and (ii) do not contain any of the other N - 3 spherical points. As before, pairs of circles that share two of the N points determine the edges of the diagram. We would like to express this problem in terms of linear components rather than circular components.

One approach is to construct a spherical analog of the formula for the planar Voronoi diagram algorithm: embed to a four-dimensional sphere, apply (spherical) inversion (with respect to a suitable point P of the four-sphere), and construct the (spherical) convex hull of the transformed points. Although this approach can actually be made to work, it does not give us a problem that involves linear components. A better approach is to embed the spherical Voronoi diagram problem (which is a spherical two-space problem) into Euclidean three-space. The circles that define the spherical Voronoi diagram determine the planes that bound the faces of the (Euclidean three-dimensional) convex hull of the N points. This convex hull can be converted readily into the *dual* of the spherical Voronoi diagram -- the spherical Delaunay diagram -- and can be constructed in O(N log N) time. Given the dual, the Voronoi diagram can be produced in only O(N) additional time.

Algorithm for Spherical Voronoi Diagram

1. Let S be a set of N > 3 points on the surface of a sphere such that no four points are co-circular. Construct the convex hull of the points of S (treating them as N points in Euclidean three-space) in O(N log N) time by the algorithm of Preparata and Hong [83].

2. For each face F_i of the convex hull there is a corresponding Voronoi point V_i on the surface of the sphere that is equidistant from the vertices of face F_i. (Actually there are two such points -- V_i and the point diametrically opposite V_i. Choose the point that is closest to the vertices of face F_i.)

3. For each pair of faces F_i and F_j that share an edge E_{ij} construct an arc of a great circle that connects points V_i and V_j. Since there are only O(N) faces and edges in the convex hull we can do this in O(N) time.

The diagram that the above algorithm constructs is the spherical nearest point Voronoi diagram because each Voronoi point V_i corresponding to face F_i is not only equidistant from the vertices of face F_i but is also closer to these three points than the other N-3 points of set S. As in the case of the planar Voronoi diagram, we have a set of circles that ensure the properties of the Voronoi diagram.

4.3. Nearest and Farthest Edge Diagrams

Drysdale and Lee [37] describe the construction of Voronoi diagrams of line segments (and other geometrical objects) in $O(N\, c^{(\log N)^{1/2}})$ time (which they later improved to $O(N (\log N)^2)$ time) and Kirkpatrick [59] has reduced this time to O(N log N). A special case of this problem is the nearest (respectively farthest) edge diagram for a convex N-gon. This diagram is a tesselation of the plane into N polygonal regions such that each region i is the set of all points nearest to (respectively farthest from) edge i of the N-gon. Figure 4-6 illustrates a nearest edge diagram.

The problem of constructing a nearest edge diagram is presented by Shamos

[89] as problem POL9 in his workbook. One interesting application is that once we have constructed the nearest edge diagram, we can solve the problem of constructing the greatest incircle of a convex polygon (problem POL16) in O(N) time. Both Preparata and Lee describe O(N log N) time solutions to this problem that are not based on geometric transforms [77, 79]. (They call this problem "Medial axis of a convex polygon.") In this section we describe an O(N log N) time solution that is based on the simple geometric transforms of (1) embedding in a higher dimension and (2) orthographic projection.

One of the main differences between the nearest-edge diagram and the others described in this chapter is that the elements defining the diagram are edges rather than points. We can, however, still express it in terms of circles; since each vertex V of the diagram is equidistant from the three nearest edges of the N-gon, V is the center of a circle that is tangent to three edges (but does not intersect or contain any of the other N - 3 edges). This suggests that we might try to apply the formula that we used for the Euclidean planar Voronoi diagram of N points; embed in Euclidean three-space, invert the edges with respect to a point P that is not in the xy plane (producing a connected set of circular arcs), and construct the convex hull of the transformed elements. This would work well if we had a fast algorithm for constructing the convex hull of a set of (connected) circular arcs in three-space.[16] There is, however, another way to embed this problem in three-space that produces a three-dimensional problem that involves only linear components.

The N circular arcs all determine circles that pass through point P. For any one of these N circles C_i there is an infinite number of spheres that pass through (all of the points of) C_i. We can thus represent the convex N-gon (in the xy plane) as a set of N spheres (in three-space) each of which passes through point P and still has one degree of freedom. How should the spheres be chosen?

---

[16] ... construction of such an algorithm, and its application to Voronoi diagrams of general sets of line ... is left as an exercise for the reader.

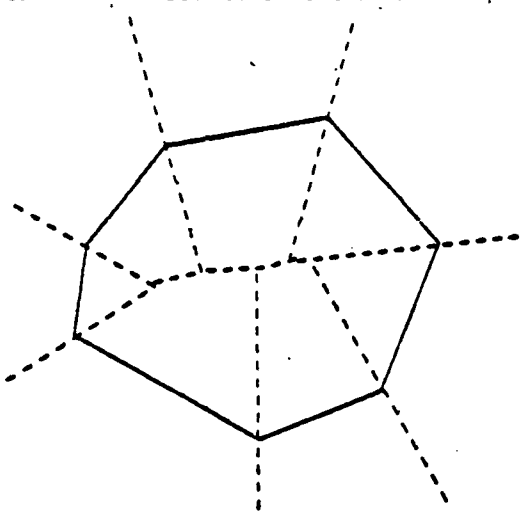


Figure 4-6: Nearest edge diagram of a convex polygon.

Since inversion maps spheres to planes and the interior of the spheres to half-spaces, we can represent the nearest edge diagram by an intersection of half-spaces. The problem of choosing the spheres now becomes a problem of choosing the half-spaces. For each half-space the degree of freedom is the angle that its boundary makes with the xy plane. Since the edges of the nearest edge diagram lie on the angular bisectors of adjacent sides of the N-gon we choose half-spaces whose boundaries meed directly above these angular bisectors. This gives us the following algorithm:

Algorithm for Nearest (Respectively Farthest) Edge Diagram for a Convex Polygon

Input: N -- number of vertices in the convex polygon. X[1:N], Y[1:N] -- x and y coordinates of vertices of the N-gon (counterclockwise order).

Output: Nearest Edge Diagram The representation is similar to the representation for an intersection of half-spaces (Section 3.1).

Time: O(N log N), Space: O(N).

1. Let the convex polygon lie in the xy plane of 3-space. For each edge e_i of the polygon construct the unique plane p_i that (a) contains that edge, (b) makes a 45 degree angle with the xy plane, and (c) lies above the polygon (rather than below it). For a nearest edge diagram, let h_i be the half-space that lies below plane p_i. For a farthest edge diagram, let h_i lie above plane p_i.

2. Intersect the N half-spaces h_i in O(N log N) time (Section 3.1).

3. Project the intersection to the xy plane. (This amounts to throwing out the z coordinates of the vertices of the intersection.)

The above algorithm does not make any distance measurements to construct the nearest (or farthest) edge diagram. Instead, it relies on the symmetry induced in Step 1 by constructing all planes p_i at the same angle (45 degrees) from the xy plane. We can generalize this to weighted distances w_i from the edges by simply letting the slopes of the planes p_i be set to the weights w_i.

4.4. Summary

We have described three types of diagrams, planar and spherical Voronoi diagrams of sets of points, and (nearest and farthest) edge diagrams for a convex polygon. Since each of these problems involved Euclidean distance between the elements defining the diagram, we found it useful to express each problem in terms of circles. We then embedded the problem in a higher dimension and, when necessary, expressed it in terms of (carefully chosen) spheres and applied

inversion to obtain an equivalent problem expressed in terms of linear components.

For the planar Voronoi diagram problem we embedded the plane into three-dimensions and applied inversion to obtain a convex hull problem. The circles defining the planar Voronoi diagram are centered at the Voronoi points and each pass through three of the N points but do not contain any of the other N - 3 points in their interiors. These circles become spheres when embedded into three dimensions and, when inverted, become the planes that bound the faces of the convex hull.

We were able to solve the spherical Voronoi diagram problem directly as a convex hull problem. This is because the circles defining the diagram pass through three of the N spherical points but do not contain any of the other N - 3 points. When the spherical Voronoi diagram problem (which is a spherical two-space problem) is embedded into Euclidean three-space, these circles become the planes that bound the convex hull of the N points.

We constructed the nearest (respectively farthest) edge diagram of a convex polygon by embedding into three-space and intersecting half-spaces. Each of the circles that define the nearest edge diagram is tangent to three of the sides of the convex N-gon and does not contain any part of the other N - 3 sides in its interior. By embedding into Euclidean three-space, applying inversion, redefining the problem in terms of spheres, and then applying inversion again we obtain a problem of intersecting half-spaces. But, as we saw in Section 3.1, the intersection of half-spaces is solved by constructing the convex hull of a set of points.

In the next chapter we will find even more uses for convex hulls.

5. Searching Tesselations

In this chapter we demonstrate how a search of a tesselation (Section 1.1.4) arises in both linear programming and computing the diameter of a set of points. Both of these problems invite the use of an orthographic projection to reduce a K-dimensional problem to a K-1-dimensional problem and furthermore provide interesting applications of the point / flat duality transform (Section 3.1.3.3). The diameter of a set of points also provides another application of the convex hull of a set of points.

5.1. Linear Programming

Linear programming is an important technique for optimizing a linear function subject to a set of several linear constraints. If there are K variables and N constraints, we may interpret each constraint as a half-space in K-space and the feasible region satisfying all of the constraints as the polytope that is the intersection of N K-dimensional half-spaces. One of the vertices of the resulting polytope is an optimal solution for the linear program. The linear programming problem is to find this vertex as quickly as possible.

The standard method of solving linear programming problems is the simplex method (and its variants) [28]. In the worst case, however, the simplex method will require exponential time [60, 55]. Kelly [58] has shown that for a model of linear programming with N relevant constraints in two variables chosen independently from a given distribution, the *expected* number of iterations is O(N). Since each iteration costs, in this case, O(N) time, the expected time for linear programming in his two-dimensional model is $O(N^2)$. In general, however, the expected-time of the simplex method has not been adequately analyzed, although empirical results indicate that it may be bounded above by a low degree polynomial in K and N [38].

There are alternatives to the simplex method. Khachian [44] has produced an algorithm for solving linear programming with integral coefficients that costs only polynomial time (in the size of the input) in the worst case. His method, however, depends strongly on the fact that the coefficients are integers. Another approach

is to intersect the N K-dimensional half-spaces to construct explicitly the feasible region and then evaluate the objective function at each vertex. For K = 2 or K = 3 variables we may construct the intersection and solve the linear programming problem in O(N log N) worst case time and (when the expected number of nonredundant constraints is $O(N^p)$ for some $p < 1$) O(N) expected-time [94, 106, 84] This is, in the worst case, better than the simplex method, which may take $O(N^2)$ time in these cases. It is not, however, generally considered advisable to construct explicitly the entire feasible region when we need only the vertex corresponding to the solution. The simplex method certainly avoids that problem. We now describe an approach toward a better solution.

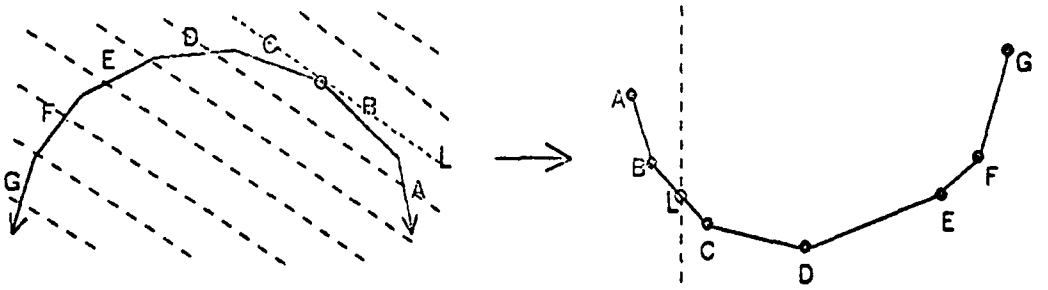


Figure 5-1: Transform of LP problem to vertical line and convex hull.

Dantzig [28] describes an alternate interpretation of the simplex method that is based on the point / flat duality transform. The polytope obtained by intersecting the N K-dimensional half-spaces transforms to the convex hull of N points in K-space and the (linear) objective function transforms to a vertical line (in the K coordinate) in K-space. (See Figure 5-1.) The linear programming problem itself is transformed to the problem of determining which face of the convex hull this vertical line intersects. When we orthographically project the convex hull to a tesselation and the vertical line to a point in K-1-space, we obtain a problem of locating a point in a tesselation. (See Figure 5-2 for an illustration of the two-dimensional case.) This does not immediately lead to a (provably) faster linear programming algorithm than the simplex method, but it does provide another way to approach the linear

programming problem.

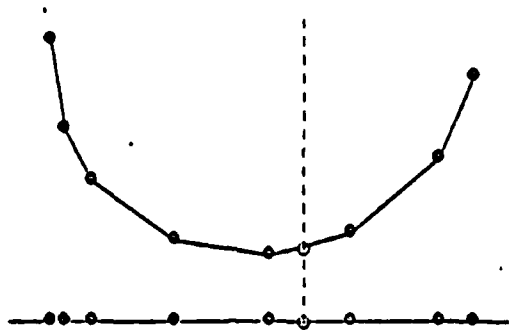


Figure 5-2: Transform of vertical line and convex hull to a point and a tesselation.

5.2. Diameter of a Set of Points

The diameter of a set of points is the distance between the two farthest points. This quantity often arises in problems of cluster analysis [51] because a set of points that are all near each other makes a better cluster than a set of points that are spread far apart. The straightforward way to determine the diameter of N points is to compute all $\theta(N^2)$ interpoint distances and return the maximum. Depending on the metric and the number of dimensions, however, there may be much faster algorithms.

For K = 1 dimension all N points lie on a line and the diameter is the distance between the points with maximum and minimum coordinate, which can easily be found in O(N) time. (In fact, $\lceil 3N/2 - 2 \rceil$ comparisons are necessary and sufficient [76]). For two or more dimensions the choice of metric is important. The L_1 or L_∞ diameter of N points in K dimensions can be easily computed in $O(2^K N)$ or O(KN) time, respectively,[17] but the Euclidean case may be more difficult. For two dimensions

[17] The diameter of a set is determined by the most extreme points in each of the directions determined by the faces of the unit "sphere". Since the unit sphere for the L_1 metric has 2^K faces, and the unit sphere for the L_∞ metric has 2K faces, we can find the extreme points in $O(2^K N)$ and O(KN) time, respectively.

the best known Euclidean diameter algorithms run in worst-case time O(N log N) [89, 91]. For three dimensions Yao [102] has produced an $O(N^{1.8})$ time algorithm and for K dimensions $O(N^{2 - \alpha(K)})$ time, where $\alpha(K) = 2^{-(K+1)}$.

We will present an O((N + K) log N) time algorithm for the three-dimensional case, where K is the number of pairs of antipodal vertices on the convex hull. To achieve this time we apply a point / flat duality combined with orthographic projection to the components of the convex hull of the set of N points to obtain a problem of locating points in an outerplanar straight-line graph. In Appendix II we describe a relationship between the Euclidean diameter and an empty-intersection problem that may lead to an Ω(N log N) time lower bound for the diameter problem. In the following sections we present O(N log N) time two-dimensional Euclidean diameter algorithms, the O((N + K) log N) time three-dimensional algorithm, and then discuss fast expected-time algorithms, approximation algorithms, higher dimensions, applications, and some unsolved problems.

5.2.1. Diameter in Two Dimensions

Shamos [91] describes an O(N log N) time algorithm for computing the diameter of N points in the plane. The algorithm that we present is essentially equivalent to his, but it is expressed so that it generalizes to a fast three-dimensional algorithm. We first present a theorem that reduces our search for the diametrical pair of points to the convex hull.

> Theorem 23: (Hocking and Young [52], p. 207) The Euclidean diameter of a set of points S is determined by two points on the convex hull of S.

If all N of the points of S are on the convex hull, then we have not reduced the size of the problem. We have, however, *simplified* it by reducing the problem of computing the diameter of a set of points to the problem of computing the diameter of a convex polygon. For our new problem we have the following theorem:

> Theorem 24: (Yaglom and Boltyanskii [100], p. 9) The diameter of a convex figure is the maximum distance between parallel lines of support of this figure.

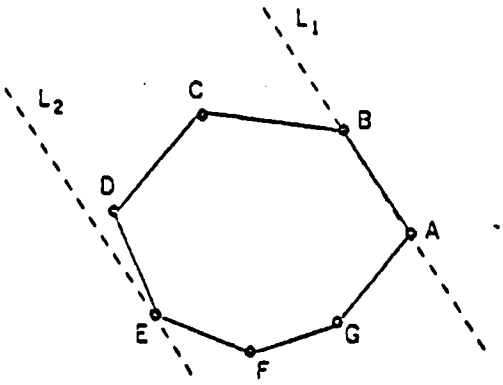


Figure 5-3: Convex hull and parallel lines of support.

In Figure 5-3 we illustrate two (parallel) lines of support L_1 and L_2. In general, a line of support passes through (at least) one boundary point of a figure and lies entirely on one side of that figure. Pairs of points (of the figure) on opposite parallel lines of support are called *antipodal points*. In Figure 5-3 points A and E and points B and E are antipodal. We are interested in the antipodal pairs of vertices determined by the lines of support for the convex hull of S because one of these pairs determines the diameter of S. Our next theorem gives a bound on the number of pairs that we will have to examine.

Theorem 25: For a convex polygon of N vertices there are only O(N) antipodal pairs of vertices.

Proof: As we rotate parallel lines of support L and M about the convex polygon, the antipodal vertices determined by L and M change only when either L or M becomes coincident with one of the N sides. We may thus generate all of the pairs of antipodal pairs of vertices by recording all antipodal pairs when either L or M contains a side of the polygon. If line L contains a side, then it passes through two vertices and similarly, its line of support M will pass through at most two vertices. (In fact, only when the polygon has parallel sides can both L and M simultaneously pass through two vertices.) There are therefore at most four antipodal pairs of vertices generated each time a line of support passes through a side of the polygon. Since there are only N sides of the polygon, there are only O(N) antipodal pairs of vertices. □

We now have enough information to outline our two-dimensional diameter algorithm:

Outline of Two-Dimensional Diameter Algorithm

1. Construct the convex hull of the N planar points.
2. Generate the O(N) antipodal pairs of vertices from the lines of support of the convex hull.
3. Compare the distances between each pair of antipodal vertices and report the maximum as the diameter.

We can construct the convex hull (Step 1) in O(N log N) time [48] and easily compare the distances between antipodal pairs of vertices (Step 3) in only O(N) time. We thus have only to determine how fast we can generate the O(N) pairs of antipodal vertices (Step 2). In the remainder of this section we show two algorithms for generating them in O(N) time, making the total time for our diameter algorithm O(N log N).

Shamos [91] describes in detail how to generate the O(N) pairs of antipodal vertices of a convex polygon in O(N) time. After finding the first pair he generates the other pairs in a counterclockwise scan about the polygon, maintaining parallel lines of support at all times. For example, in Figure 5-3 line L_1 passes through vertices A and B and parallel line of support L_2 passes through vertex E. We can rotate L_1 counterclockwise about vertex B and L_2 about vertex E until either L_1 contains side BC or L_2 contains side EF. We determine which of the two possibilities occurs first by comparing the slopes of sides EF and BC. In this case line L_2 will meet side EF before line L_1 meets BC because the slope of EF is less than the slope of BC. This means that vertices B and F are antipodal and that we will begin rotating L_2 about vertex F rather than E. We continue this procedure until the parallel lines of support L_1 and L_2 have traversed the entire convex polygon (and have thus generated all of the antipodal pairs of vertices).

We can modify Shamos' algorithm so that it generalizes easily to the three-dimensional case. The important feature that we extract from his algorithm is that when we perform the O(N) time scan around the convex hull, the only comparisons that we must make are comparisons of the *slopes* of the sides of the convex hull. That is, the x and y coordinates of the vertices do not matter since we compare only the slopes of the sides of the polygon. This insight leads to a one-dimensional interpretation of the (originally) two-dimensional problem.

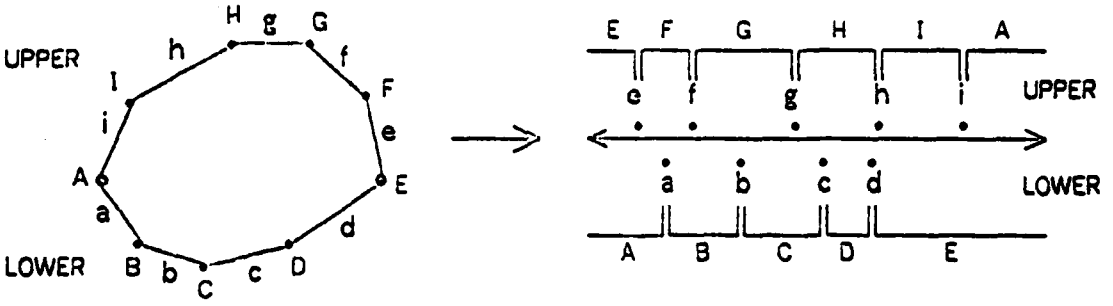


Figure 5-4: Transform of a convex hull to a line.

We illustrate the transform in Figure 5-4. The first step is to divide the convex polygon into two parts, UPPER and LOWER. This ensures that when one of two parallel lines of support meets the polygon at an UPPER side, the other line of support meets it a LOWER side. We may define the transform of an UPPER side of the convex hull as the slope of the UPPER line of support that contains it. We may also transform an UPPER vertex V to the set of slopes of all UPPER lines of support that pass through V. The transform for LOWER sides and vertices is similar. Furthermore, since we consider the leftmost and rightmost vertices (A and E in Figure 5-4) to lie in both the UPPER and LOWER sets, they each have both an UPPER and a LOWER transform. We now describe the transform algebraically.

The transform maps the UPPER set of sides of the convex hull to a set of points on the line and the LOWER set of sides to another set of points on the line. The mapping is simply a point / flat duality followed by an orthographic projection

$$y = mx + b \rightarrow (m,b) \rightarrow (m) \qquad (14)$$

where "y = mx + b" is the line determined by a side of the convex hull and "(m)" is the one-dimensional point to which the side maps. The transform of a vertex V of the convex hull is an interval on the line. If V is the intersection of two UPPER (or two LOWER) sides that determine the lines $y = m_1x + b_1$ and $y = m_2x + b_2$ then the transform of V is the interval between (one-dimensional) points (m_1) and (m_2). This is because all UPPER (or LOWER) lines of support at V must have a slope between (m_1) and (m_2). If V is a leftmost or rightmost point, then the set of slopes of the UPPER (or LOWER) lines of support at V is an infinite interval on the line. For example, in Figure 5-4 the UPPER transform of vertex E is the interval $(-\infty,e]$ and the LOWER transform of vertex E is the interval $[d,\infty)$, where "e" is the slope of side e and "d" is the slope of side d.

The transform gives us all the information we need to generate the O(N) pairs of antipodal vertices. For example, in Figure 5-4, if L_1 is the line determined by side a, then the parallel line of support L_2 passes through the vertex F. Equivalently, the transform maps side a to point a and vertex F to interval F such that point a lies inside the interval F. Since side a is bounded by vertices A and B, we have the antipodal pairs of vertices (A,F) and (B,F). We can generate all O(N) pairs of antipodal vertices by finding which intervals contain the N points a, b, c, etc.

Theorem 26: Given a convex polygon of N sides, we can generate the O(N) pairs of antipodal vertices in O(N) time.

Proof: When we generate the UPPER and LOWER sets of points on the line (Equation 14), they will be in sorted order because the slopes of the sides of the convex hull are already sorted. We can thus easily scan the two sets to determine which interval each point lies in and ultimately generate the O(N) antipodal pairs of vertices in O(N) time. □

In summary, we have

Theorem 27: We can compute the diameter of N planar points in O(N log N) worst-case time.

Proof: We first construct the convex hull in O(N log N) ([48]) time and then compare the O(N) antipodal pairs of vertices in O(N) time. To

generate the antipodal pairs we may use either the scan around the convex hull of Shamos [91] or first transform the sides of the convex hull to points on a line and then perform an equivalent scan of those points.

□

5.2.2. Diameter in Three Dimensions

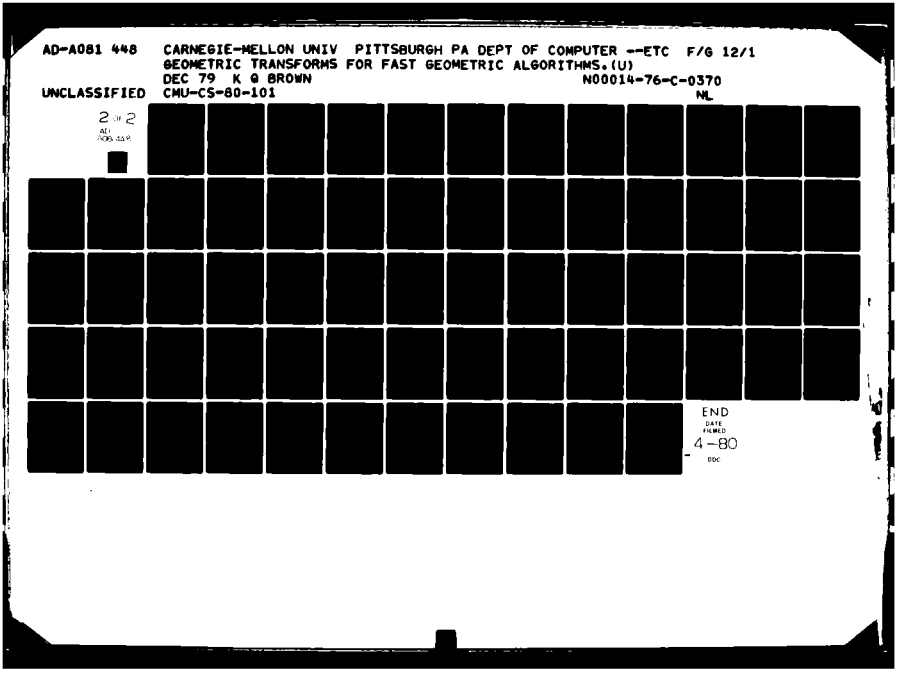
Many features of our algorithm for the diameter of a two-dimensional set of points extend to three dimensions. We first construct the convex hull of the N points in O(N log N) time (by the algorithm of Preparata and Hong [83]) to enable us to solve the diameter of the set of points as the diameter of a convex hull. To find the diameter of the convex hull we then generate the set of antipodal pairs of vertices, one pair of which determines the diameter. In this section we show how to generate the K pairs of antipodal vertices in O((N + K) log N) time and that we can thereby compute the diameter of N points in three-space in O((N + K) log N) time.

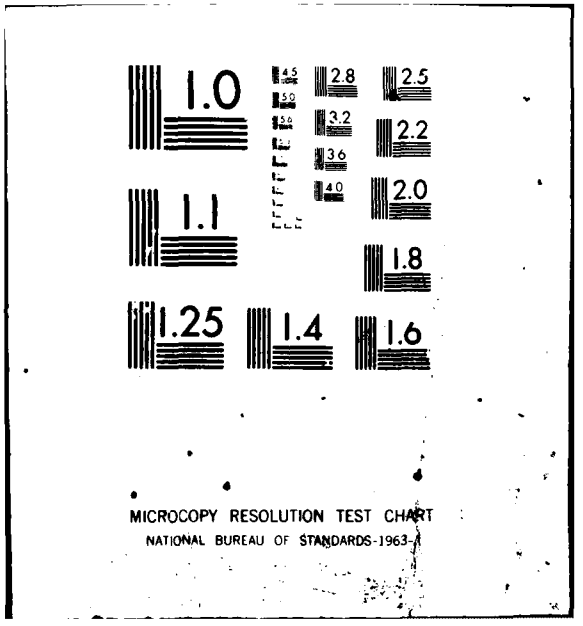
In the plane we used the concept of line of support to generate the O(N) pairs of antipodal vertices. In three dimensions the corresponding concept is *plane* of support. For each face of the convex hull, say face PQR, the plane of suppor passes through a vertex W of the convex hull. This generates the antipodal pairs of vertices (P,W), (Q,W), and (R,W). Although the convex hull has only O(N) faces, in the worst case there may still be $\Omega(N^2)$ pairs of antipodal vertices.[18] When the number of pairs K is less than $\Omega(N^2)$, though, it is not obvious how to generate them in less than $\Omega(N^2)$ time.

Our first step toward generating the antipodal vertices is to divide the faces of the convex hull into the two sets UPPER and LOWER. (The plane that contains an UPPER face lies above the convex hull and the plane that contains a LOWER face lies below the convex hull.) This division has the property that if a plane L contains a face of the UPPER set, then the plane of support M that is parallel to L passes

---

[18] For example, there are $\Omega(N^2)$ antipodal pairs if N/2 of the points are of the form $(0, \cos\theta_i, \sin\theta_i)$ and N/2 of the points are of the form $(\cos\theta_i, 0, -\sin\theta_i)$ where $\theta_i = \pi/2*(1 \pm i/N)$.


AD-A081 448 CARNEGIE-MELLON UNIV PITTSBURGH PA DEPT OF COMPUTER --ETC F/G 12/1
GEOMETRIC TRANSFORMS FOR FAST GEOMETRIC ALGORITHMS.(U)
DEC 79 K Q BROWN N00014-76-C-0370
UNCLASSIFIED CMU-CS-80-101 NL
2 OF 2
END
DATE FILMED
4-80
DDC


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

through a vertex of the LOWER set. Similarly, a face of the LOWER set determines a parallel plane of support that passes through a vertex of the UPPER set. (We consider the vertices on the boundary of the UPPER and LOWER sets to belong to both sets.) We next show how we can transform the UPPER and LOWER sets to find the antipodal vertices quickly.

5.2.2.1. Transform in Three Dimensions

To generate the O(N) pairs of antipodal vertices we use an extension of the geometric transform that we used for the two-dimensional problem. In two dimensions our choice of transform was motivated by the fact that the search for lines of support involves only comparisons of the *slopes* of the sides of the convex hull. Similarly, in three dimensions, the search for planes of support involves comparisons of the slopes of the faces of the convex hull. Our transform for the faces, edges, and vertices of the convex hull all follow the same schema:

> The UPPER transform of a component of the convex hull is the set of slope-pairs of all UPPER planes of support that contain it. Similarly, the LOWER transform is the set of slope-pairs of all LOWER planes of support that contain it.

Since only one plane of support contains a face of the convex hull, a face maps to one point. An edge is contained by a set of planes of support with one degree of freedom so an edge maps to an interval of a line. Finally, a vertex of the convex hull is contained by a set of planes of support with two degrees of freedom so a vertex maps to a planar region. We now present algebraic descriptions of the transforms of faces, edges, and vertices of the convex hull.

Transform of a Face

Let $z = ax + by + c$ be the plane determined by a face F of the convex hull. The transform of face F is simply the pair of slopes (a,b), a point in the ab plane. We may alternately view this transform as a combination of the point / flat duality and orthographic projection:

$$z = ax + by + c \rightarrow (a,b,c) \rightarrow (a,b). \qquad (15)$$

We next describe the transform of an edge and a vertex. Since the UPPER and LOWER transforms are so similar, we present only the UPPER transform for each case.

UPPER Transform of an Edge

Suppose that two faces L and M of the convex hull have an UPPER edge E_{LM} in common. Let the two planes L and M determined by these faces be written

$$\begin{aligned} z &= a_L x + b_L y + c_L, \text{ and} \\ z &= a_M x + b_M y + c_M. \end{aligned} \tag{16}$$

We can write the line where planes L and M meet in parametric form as

$$(P_x,P_y,P_z) + u \begin{vmatrix} \bar{I} & \bar{J} & \bar{K} \\ a_L & b_L & -1 \\ a_M & b_M & -1 \end{vmatrix}, \quad u \in \text{Reals} \tag{17}$$

where $P = (P_x,P_y,P_z)$ is some point in both planes and $\bar{I}$, $\bar{J}$, and $\bar{K}$ are unit vectors parallel to the x, y, and z axes, respectively. For example, if the line intersects the xy-plane we can choose P as

$$P = (P_x,P_y,P_z) = - \begin{bmatrix} a_L & b_L & 0 \\ a_M & b_M & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} c_L \\ c_M \\ 0 \end{bmatrix}.$$

A plane $z = ax + by + c$ contains the line of Equation (17) iff

$$\begin{vmatrix} a & b & -1 \\ a_L & b_L & -1 \\ a_M & b_M & -1 \end{vmatrix} = 0 \quad \text{and} \quad P_z = aP_x + bP_y + c. \tag{18}$$

Since the transform of E_{LM} is limited to UPPER planes *of support* that contain E_{LM}, we must restrict the solutions of Equation (18) to an interval defined by the two points (a_L, b_L, c_L) and (a_M, b_M, c_M). The transform of edge E_{LM} is the projection of

an interval of this line to the xy (ab) plane. We summarize our results in the following theorem:

Theorem 28: Let the planes L and M of Equation (16) be determined by two faces of a convex polyhedron that meet at an UPPER edge E_{LM}. The UPPER transform of edge E_{LM} is the set of points (a,b) on the line

$$(b_M - b_L)a - (a_M - a_L)b = a_L b_M - a_M b_L \qquad (19)$$

that lie in an interval determined by the points (a_L, b_L) and (a_M, b_M). If L and M are both UPPER planes then the interval lies between the two points. If L is UPPER but M is LOWER then the interval is the ray from point (a_L,b_L) that does *not* include point (a_M,b_M). (Similarly when L is LOWER and M is UPPER.)

Proof: The line of Equation (19) is from Equation (18). Since the set of UPPER planes of support that contain edge E_{LM} is connected, the UPPER transform of edge E_{LM} is an interval of this line. Since the planes L and M of Equation (16) contain edge E_{LM}, the points (a_L,b_L) and (a_M,b_M) must be on the UPPER transform of edge E_{LM} iff planes L and M are UPPER faces of the convex hull. Furthermore, since the faces L and M are the extreme limits that a plane of support can be rotated about edge E_{LM}, point (a_L,b_L) must be an endpoint of the interval if plane L is an UPPER plane of support and (a_M,b_M) must be an endpoint of the interval if plane M is an UPPER plane of support. From these conditions it follows that if both L and M are UPPER planes of support, then the interval must be between the two points. If L is an UPPER plane of support but not M, then the interval is a ray starting from point (a_L,b_L) that does not contain point (a_M,b_M). Similary if L is LOWER and M is UPPER. □

UPPER Transform of a Vertex

Suppose that H faces meet at a vertex $V = (V_x,V_y,V_z)$ on the UPPER part of the convex hull. Number these faces in counterclockwise order so that the edges that meet at vertex V are $E_{12}, E_{23}, \ldots, E_{H1}$. (If vertex V lies on the boundary of the UPPER and LOWER parts then include only those edges in the UPPER part.) Let the planes determined by the H faces be

$$z = a_i x + b_i y + c_i, \quad i = 1, \ldots, H.$$

If ray r_i is the ray that originates at point V and points down edge $E_{i,i+1}$ then

$$r_i = (V_x, V_y, V_z) + u\,(\alpha_i, \beta_i, \gamma_i), \quad u \geq 0 \qquad (20)$$

where

$$(\alpha_i, \beta_i, \gamma_i) = \alpha_i \bar{I} + \beta_i \bar{J} + \gamma_i \bar{K} = \begin{vmatrix} \bar{I} & \bar{J} & \bar{K} \\ a_i & b_i & -1 \\ a_{i+1} & b_{i+1} & -1 \end{vmatrix} \times \text{SGN} \begin{vmatrix} a_i & b_i & -1 \\ a_{i+1} & b_{i+1} & -1 \\ a_{i+2} & b_{i+2} & -1 \end{vmatrix}. \qquad (21)$$

(Note that the subscripts "i+1" and "i+2" are to be taken modulo H.) The first determinant in Equation (21) determines the line that ray r_i lies in and the second determinant determines which of the two possible directions r_i should point. We may now characterize the set of UPPER planes of support of UPPER vertex V.

Theorem 29: Let $V = (V_x, V_y, V_z)$ be an UPPER vertex of a convex polyhedron at which the UPPER rays r_i determined by V satisfy Equation (20). A plane $z = ax + by + c$ is an UPPER plane of support at vertex V iff

$$V_z = aV_x + bV_y + c \qquad (22)$$

and

$$\gamma_i \leq a\alpha_i + b\beta_i, \quad i = 1, \ldots, H. \qquad (23)$$

Proof: A plane $z = ax + by + c$ is an UPPER plane of support at vertex V iff it passes through V and remains above all the rays r_i. Equation (22) requires the plane to pass through V. The plane remains above the H rays iff

$$V_z + u\gamma_i \leq a(V_x + u\alpha_i) + b(V_y + u\beta_i) + c, \quad \forall u \geq 0. \qquad (24)$$

By subtracting Equation (22) and dividing by u we obtain the inequality (23). Conversely, if Equations (22) and (23) are satisfied, we easily derive Equation (24) by multiplying (23) by $u \geq 0$ and adding to (22). □

For the transform of UPPER vertex V we are interested only in the slopes of the UPPER planes of support at V. By Theorem 29 we have

Theorem 30: Let V be an UPPER vertex of a convex polyhedron at which the UPPER rays r_i determined by V satisfy Equation (20). The UPPER transform of V is a convex polygonal region of the ab plane determined by the inequalities

$$\gamma_i \leq a\alpha_i + b\beta_i, \quad i = 1, \ldots, H.$$

5.2.2.2. Algorithm for Generating Antipodal Vertices

We have just seen how to transform the faces of a convex polyhedron to points in the plane, edges to edges in the plane, and vertices to convex polygonal regions. Combining these we can transform both the UPPER and LOWER parts of the convex hull to outerplanar straight-line graphs in the plane, as illustrated in Figure 5-5. We now present the property of these graphs that allows us to find efficiently the K pairs of antipodal vertices.

Let V be an UPPER vertex of a convex polyhedron. The UPPER transform of V is a convex polygonal region V' of the ab plane. By definition of V', any UPPER plane that passes through vertex V transforms to a point in V' iff it is an UPPER plane of support. Any LOWER plane of support that maps to a point in V' is therefore a parallel plane of support for some UPPER plane of support that passes through V. If a LOWER vertex W maps to a region W' such that V' and W' overlap, then V and W share parallel planes of support and are therefore antipodal.

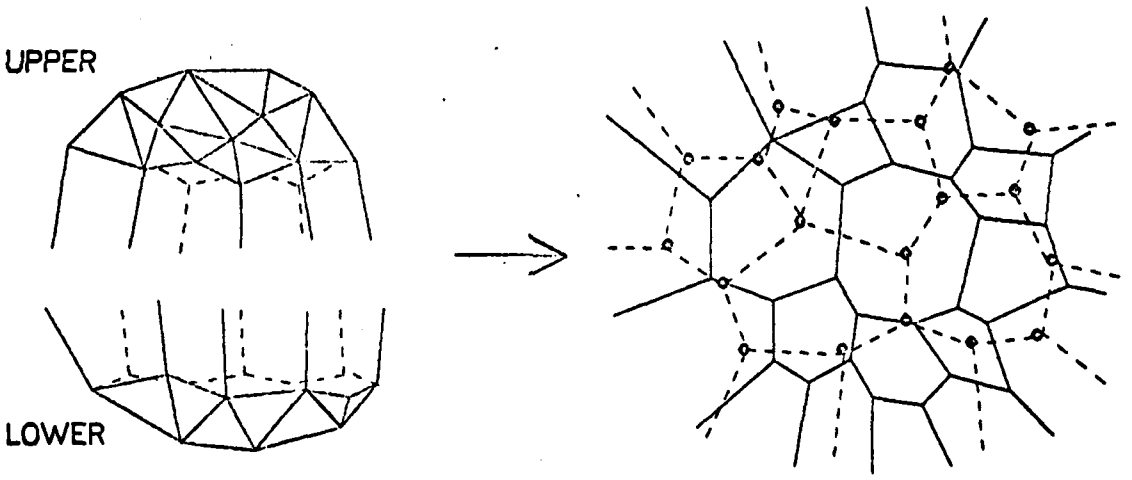


Figure 5-5: Search for planes of support maps to locating overlapping regions.

We have reduced the problem of computing the diameter of N points in three-space to the problem of finding intersections of the regions of two outerplanar

straight-line graphs. We thus have the following outline for a three-dimensional diameter algorithm:

Outline of Three-Dimensional Diameter Algorithm

1. Construct the convex hull of the N points.
2. Divide the convex hull into the two parts UPPER and LOWER.
3. Map both the UPPER and LOWER parts to outerplanar straight-line graphs by the transform described above.
4. Find all of the pairs of regions in the UPPER and LOWER graphs that overlap. Update the maximum distance between pairs of antipodal vertices.
5. Report the maximum distance measured as the diameter.

We can construct the convex hull (Step 1) in O(N log N) time [S3] and in O(N) time partition the hull into the sets UPPER and LOWER (Step 2) and transform these sets to outerplanar straight-line graphs (Step 3). Since there are only O(N) pairs of antipodal vertices, Step 5 requires only constant time. This leaves Step 4, finding the overlaps between the UPPER and LOWER outerplanar graphs. We now describe how to conduct this search in O((N + K) log N) time, where K is the number of pairs of antipodal vertices.

Assume that there are O(N) faces on both the UPPER and LOWER parts of the convex hull. The subproblem of Step 4 is thus to find the overlaps among two sets of O(N) convex regions (with a total of O(N) edges). Two regions overlap iff (1) a vertex of one region is contained in the other region or (2) an edge of one region intersects an edge of another region. There exist algorithms that solve these two cases separately; we only need to combine them.

We can solve the first case, determing inclusion of the vertices in planar regions, by a number of algorithms. Lee and Preparata [G6] describe how we can locate a point in the correct region in $O(\log^2 N)$ time, with only O(N log N) preprocessing time.

We can thus locate the O(N) points in $O(N \log^2 N)$ time. Lipton and Tarjan [70] have improved the query time to O(log N), yielding an O(N log N) time algorithm for the O(N) points while using only O(N) storage, but the "constant factor" of their algorithm is very large. Preparata [80] has produced a practical algorithm that costs only $6\lceil \log N \rceil$ comparisons for each query but requires O(N log N) storage and preprocessing time. Since we want to locate a set of points together rather than just one at a time, however, we can use Preparata's O(K log K) + O(N) + O(K log N) time algorithm [82] or Lee and Yang's O((N + K) log(N + K)) time algorithm [68] for locating a set of K points in a straight-line planar graph of N vertices, giving us an O(N log N) time algorithm for locating O(N) points.

The second case, finding all K intersections of the edges of the two graphs, can be solved in O((N + K) log N) time and O(N) storage by Brown's modification [24] of Bentley and Ottmann's [12] intersection algorithm (which is itself a modification of Hoey's algorithm for determining if any of N line segments intersect [94, 91]).

The algorithm below finds all of the edge intersections and all of the vertex inclusions in O((N + K) log N) time and O(N) storage. It finds the edge intersections by Brown's algorithm [24] and the regions containing the vertices by maintaining two order relations (R_U and R_L) for the two outerplanar graphs. Since the extra time required to maintain R_U and R_L is only O(N log N) and the storage is only O(N), the total time and storage bounds are the same as for the edge intersection algorithm. This algorithm uses several simple data structures and functions:

- G_U and G_L - two outerplanar straight-line graphs representing the transforms of the UPPER and LOWER parts of the convex hull of the three-dimensional points,

- Nextint[i] = next detected intersection point for segment i (that is to the right of the current x coordinate of the left-to-right scan),

- Q - a queue of (<intersection-point or endpoint>,) pairs, sorted by the x coordinates of the intersection points (or endpoints),

- R_U [R_L] - an order relation of edges from G_U [G_L] (evaluated at the current x coordinate of the left-to-right scan),

- R - an order relation of line segments for both G_U and G_L combined (evaluated at the current x coordinate of the left-to-right scan),

- Regions[i] = ordered pair (Regions[i].above, Regions[i].below) of the regions above and below edge i in the graph (G_U or G_L) of edge i,

- Point[i] = (x_i, y_i, z_i) of the three-dimensional point corresponding to region i of G_U or G_L,

- Vertices[i] = (three) vertices of the convex hull that determine the face that maps to vertex i of G_U or G_L.

- Insert(P,A,Q) - inserts (P,A) into the queue Q, where P is the left or right endpoint of segment A or the intersection of segment A and another segment.

- Delete(P,A,Q) - deletes (P,A) from the queue Q. Exception: If P.x = ∞ the request is ignored.

- Insert(S,R) [Delete(S,R)] - inserts [deletes] segment S in [from] order relation R (where the order is evaluated at the x coordinate of the left endpoint of S).

- Above(P,R) [Below(P,R)] - returns the segment above [below] point P in order relation R (where the order is evaluated at P.x),

- Insert(S,R_U,R_L) [Delete(S,R_U,R_L)] - inserts [deletes] segment S in [from] the order relation (R_U or R_L) to which S belongs (where the order is evaluated at the x coordinate of the left endpoint of S),

- Above(P,R_U,R_L) [Below(P,R_U,R_L)] - returns the segment above [below] point P in the order relation (R_U or R_L) to which P does <u>not</u> belong (where the order is evaluated at P.x).

- Pairs(L,M,D) - computes the distances between all pairs of points (i,j) such that i ∈ L and j ∈ M. If any of these distances are greater than D, then D is set to the new maximum.

Algorithm for Search Step of Diameter Algorithm

```
proc Inter(A,B)
! Implement the modified insertion rule of [23] for segments A and B.;
   P ← Intersection(A,B);
   L ← NextInt[A];  M ← NextInt[B];
   if P.x < L.x then
      Delete(L,A,Q); Insert(P,A,Q); NextInt[A] ← P;
   if P.x < M.x then
      Delete(M,B,Q); Insert(P,B,Q); NextInt[B] ← P;


! Initialization;
   Q ← { all pairs (P,i) where P is a left or right endpoint of segment i,
         sorted by the x coordinates P.x }
   R ← R_U ← R_L ← φ;  ! order relations for the edges of G_U and G_L;
   NextInt[i].x ← ∞ for all segments i;
   Diam ← 0;
```

```
while Q ≠ φ do
   (P,S) ← (next (point,segment) pair on Q);  ! X coord of scan becomes P.x;
   T ← (other segment intersecting at P if the next pair on Q is (P,T));
   If P is the left endpoint of segment S then
      Insert(S,R);
      A ← Above(P,R);  B ← Below(P,R);
      If A intersects S then Inter(A,S);
      If B intersects S then Inter(B,S);
      Pairs(Point[Regions[Above(P,R_U,R_L)].below], Vertices[P], Diam);  ! Three pairs;

   Else if P is the right endpoint of segment S then
      A ← Above(P,R);  B ← Below(P,R);
      Delete(S,R);
      If A intersects B then Inter(A,B);
      Pairs(Point[Regions[Above(P,R_U,R_L)].below], Vertices[P], Diam);  ! Three pairs;

   Else   ! P is an intersection of segments S and T.;
      Report(P);
      NextInt[S].x ← NextInt[T].x ← ∞;
      Reverse(S,T,R);  ! Let S become the top segment.;
      A ← Above(P,R);  B ← Below(P,R);
      If A intersects S then Inter(A,S);
      If B intersects T then Inter(B,T);
      Pairs(Regions[S], Regions[T], Diam);  ! Compare four pairs.;
```

We have just seen how to find all of the O(K) overlaps of regions of two outerplanar straight-line graphs of size O(N) in O((N + K) log N) time and O(N) storage. But, as shown above, this implies

> Theorem 31: The diameter of N points in three-space can be computed in O((N + K) log N) time, where K is the number of pairs of antipodal vertices on the convex hull of the N points.

5.2.3. Refinements, Extensions, Related and Unsolved Problems

We have concentrated only on worst-case two- and three-dimensional algorithms for computing the Euclidean diameter exactly. In this section we briefly describe results for fast expected-time, approximation, and higher-dimensional algorithms, open problems and an application to Chebyshev regression.

1. One of the major open problems for the Euclidean diameter is proving a nontrivial lower bound. In Appendix II we show that a diameter algorithm can solve an empty-intersection problem for which an $\Omega(N \log N)$ time lower bound has been proven for a weak model of computation, but there is still no $\Omega(N \log N)$ time lower bound for a model of computation strong enough to construct a convex hull in $O(N \log N)$ time. Shamos [92] conjectures that the diameter problem has a worst-case lower bound of $\Omega(N \log N)$ time for any metric whose circle has a continuously-turning tangent (such as the Euclidean metric), but that if the circle has only a discretely-turning tangent (such as the L_1 or L_∞ metrics) then we can compute the diameter in $O(N)$ time.

2. In our two-dimensional diameter algorithm the most expensive step is the construction of the convex hull of the N points in $O(N \log N)$ worst-case time. A fast *expected-time* convex hull algorithm leads to a fast diameter algorithm [91]. For example, if the expected number of vertices on the convex hull is only $O(N^p)$ for some $p < 1$, then we may construct the convex hull in $O(N)$ expected-time [16] and therefore compute the diameter of N points in $O(N)$ expected-time.

   Our three-dimensional algorithm, however, takes not only $O(N \log N)$ worst-case time to construct the convex hull, but also $O((N + K) \log N)$ time to generate the K pairs of antipodal vertices. If the expected number of vertices on the convex hull is only $O(N^p)$ for some $p < 1$ and the expected number of pairs of antipodal vertices is $\bar{K}$, then we may compute the diameter in $O(N + \bar{K} \log N)$ expected-time. An obvious open problem is to prove bounds for $\bar{K}$, as a function of N for interesting distributions of points. (Conjecture: Let the straight-line planar graphs that the UPPER and LOWER parts of the convex hull map to be G_U and G_L and let the regions of G_U be U_i and the regions of G_L be L_i. For any bounded region m of G_U or G_L let $R(m,\theta)$ be the aspect ratio of m -- the ratio of height to width -- when m is rotated an angle θ. For any unbounded region m with sides along rays r and s, let $R(m,\theta)$ be the aspect ratio for any (rotated) isosceles triangle with its two equal sides on lines r and s. The number of overlapping regions K is $O(S^{1/2}N)$, where $S = \max(i,j,\theta)\ R(U_i,\theta) / R(L_j,\theta)$. This type of bound arises in the maximum overlap of N rectangular regions [22].)

   In D dimensions we can construct the convex hull of N points in $O(N)$ expected-time if the D coordinates of the points are independently distributed [10] [30]. In this case the expected number of vertices on the convex hull is only $O(\log^{D-1}N)$ [10] and we can then compute the diameter in only $O(\log^{2(D-1)}N)$ more expected-time by the

brute-force method of comparing all pairs of vertices.

3. The best known worst-case three-dimensional Euclidean diameter algorithm is that of Yao [102], which runs in $O(N^{1.8})$ time. Yao has also produced a D-dimensional Euclidean diameter algorithm that runs in $O(N^{2-\alpha(D)})$ time where $\alpha(D) = 2^{-(D+1)}$. (This algorithm does not involve construction of a convex hull because a convex hull in four or more dimensions may have $\Theta(N^2)$ edges ([49], p.193).)

4. Yuval's O(N log N) time two-dimensional Chebyshev regression algorithm [91] relies on a scan of a convex hull with lines of support. We may apply our O((N + K) log N) time algorithm for locating planes of support of a convex polyhedron to obtain an O((N + K) log N) time three-dimensional Chebyshev regression algorithm.

5. We may *approximate* the Euclidean diameter of a planar set of N points within a factor of $1 + \epsilon$ in worst-case time $O(N + 1/\epsilon)$ [7] (Section 6.1). (Shamos and Yuval [95] have previously described a technique that leads to an $O(N/(\epsilon^{1/2}))$ time approximation algorithm.) Bentley, Faust and Preparata [7] describe a D-dimensional ϵ-approximate algorithm that runs in $O(N + (1/\epsilon)^{2(D-1)})$ time. Open Problem: Construct a faster D-dimensional ϵ-approximate diameter algorithm or prove a nontrivial lower bound for the D-dimensional ϵ-approximate diameter problem.

5.3. Summary

For both of the problems treated in this chapter (linear programming and the Euclidean diameter of a set of points) we have found that an apparently D-dimensional problem can be expressed as a D-1-dimensional problem. This is because each problem can be expressed as a search for a flat (or a pair of flats) in which only the D - 1 slopes (but not the intercept) are important. The transform is chosen not only to reduce the dimensionality of the problem but also to represent it in a form for which there are already fast algorithms. The point / flat duality transforms the search of flats into a search of points and orthographic projection removes the unneeded coordinate.

6. Miscellaneous Problems and Techniques

In this chapter we present two problems and solutions that do not fit into any of the categories of the previous chapters. The techniques applied to the first problem (finding the approximate diameter of a set of planar points) provide a contrast to the techniques for the exact diameter in the previous chapter because the emphasis is on quickly producing an approximate convex hull rather than on a fast search of an exact convex hull. Our solution to the second problem (fitting points on a hemisphere) introduces a new transform called gnomonic projection that has the property of mapping great circles on a sphere to straight lines in the Euclidean plane.

6.1. Approximate Diameter of Points in Two Dimensions

We presented an O(N log N) time algorithm for computing the exact Euclidean diameter of N planar points in Section 5.2. In this section we describe two O(N) time algorithms for approximating the Euclidean diameter of N planar points within a relative factor of $(1+\epsilon)$.[19] To achieve the faster time we must use radically different techniques. Whereas the exact diameter algorithm used the point / flat duality and orthographic projection to obtain a problem of locating points within a tesselation, the first approximation algorithm uses rotation to define a metric whose unit circle is a regular polygon and the second approximation algorithm extends the ideas of the first by defining a transform based on a "pie-slice" diagram and use of the floor function.

6.1.1. First Approximate Diameter Algorithm

Shamos and Yuval [95] described how to approximate the Euclidean metric with a metric whose unit circle is a regular polygon -- the distance between two points P and Q is the width of the smallest regular K-gon (of a particular orientation) that contains both P and Q. (Figure 6-1 illustrates the case for K = 8 sides.) This

---

[19] The source of this problem is a contest held by M.I. Shamos at Carnegie-Mellon University during Fall 1978.

approximation has the property that as the number of sides of the regular polygon increases the approximation improves. Our problem is to find a function K(ϵ) such that we can approximate the diameter within a factor of 1 + ϵ by using a regular polygon metric of K(ϵ) sides.

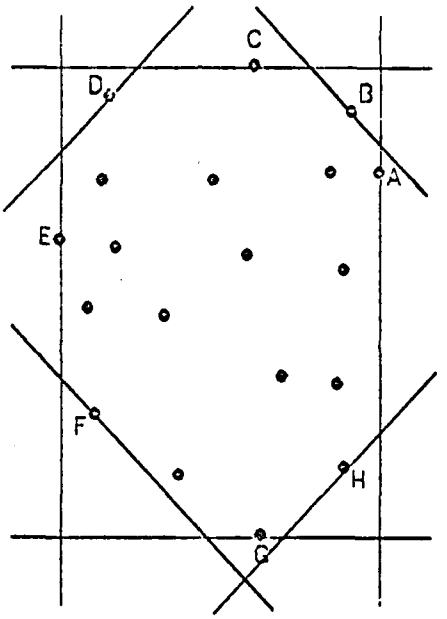


Figure 6-1: Points A through H determine the "octagon" metric diameter

Theorem 32: Let $DIAM_K$ be the diameter of a planar set of points S in a regular K-gon metric and let DIAM be the Euclidean diameter of S. If

$$K \geq \lceil \pi / \sec^{-1}(1 + \epsilon) \rceil$$

then

$$DIAM_K \leq DIAM \leq (1 + \epsilon)\, DIAM_K.$$

Proof: The worst cases are pictured in Figures 6-2 (a) and (b). In Figure 6-2 (a) the Euclidean diameter is determined by points P and Q but the K-gon diameter is determined by points T and U and points V and W. Letting D(A,B) denote the Euclidean distance between two points A and B, we define

$$r = D(T,U) = D(V,W) \quad \text{and} \quad R = D(P,Q).$$

Since the metric is based on a regular K-gon, the angle α between lines L_T and L_V and between lines L_U and L_W is

$$\alpha = 2\pi / K.$$

The minimum possible K-gon diameter r is achieved when points T and U are placed so that line TU is perpendicular to lines L_T and L_U and points V and W are placed so that line VW is perpendicular to lines L_V and L_W. From these conditions it follows that

$$r = R \cos(\alpha/2).$$

Since $\epsilon = (R - r) / r$, we have

$$K \geq \lceil \pi / \sec^{-1}(1 + \epsilon) \rceil.$$

In Figure 6-2 (b) the Euclidean diameter is determined by points P and Q but the K-gon diameter is determined by points P and B and points P and C. Letting $r = D(P,B) = D(P,C)$ and $R = D(P,Q)$ we have

$$\epsilon = \frac{R - r}{r} = \frac{\sqrt{r^2 - s^2} + s\tan(\alpha/2) - r}{r} = \sqrt{1 - (s/r)^2} + (s/r)\tan(\alpha/2) - 1,$$

which is maximized at $s/r = \sin(\alpha/2)$, so that

$$\epsilon \leq \cos(\alpha/2) + \sin(\alpha/2)\tan(\alpha/2) - 1 = \sec(\alpha/2) - 1.$$

Since $\alpha = 2\pi / K$, we have

$$K \geq \lceil \pi / \sec^{-1}(1 + \epsilon) \rceil.$$

□

Theorem 33: If $K = \lceil \pi / \sec^{-1}(1 + \epsilon) \rceil$, then as $\epsilon \to 0$, $K \to \pi / (2\epsilon)^{1/2}$.

Proof: Since $\cos(x) = 1 - x^2/2 + \ldots$, the result follows. □

First Approximate Diameter Algorithm

1. Let the number of sides of the regular K-gon metric be

$$K = \lceil \pi / \sec^{-1}(1 + 2\epsilon) \rceil.$$

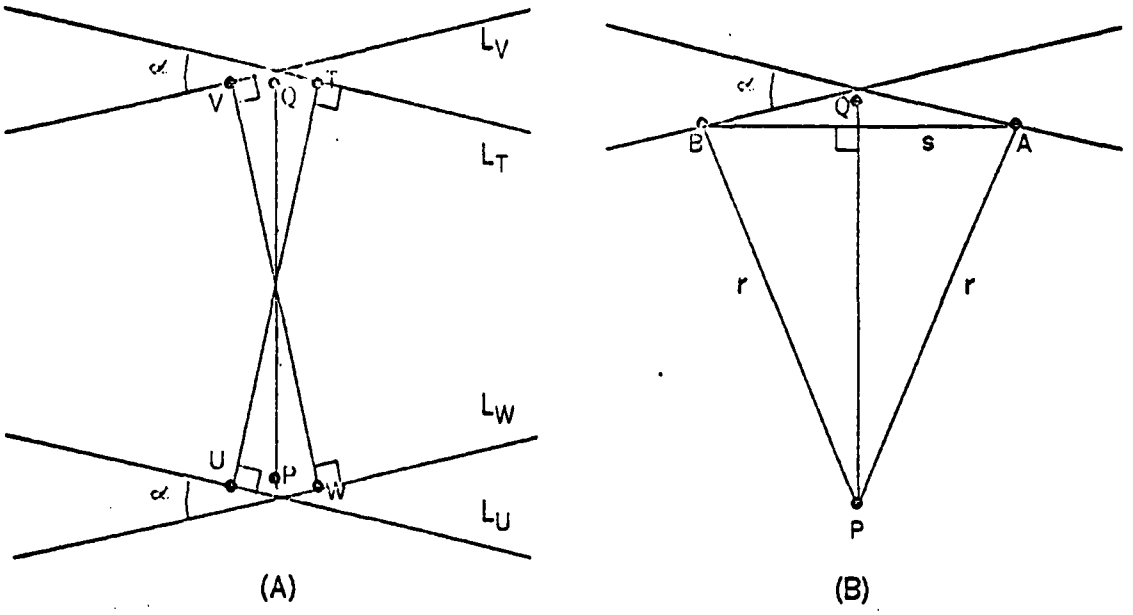


Figure 6-2: Worst cases for K-gon diameter.

2. For each of the directions (angles) 0, $2\pi/K$, $4\pi/K$, $6\pi/K$, . . . $2(K-1)\pi/K$ find the most extreme of the N points. (This is equivalent to repeatedly rotating the N points and finding the point farthest to the right.)

3. Find the (exact) diameter of the (at most) K points determined in Step 2 and multiply it by $(1 + \epsilon)$. This can be done in O(K) time by a Graham [48] scan to construct the convex hull and a Shamos [89] scan to compute the diameter.

Theorem 34: The Euclidean diameter of a set of N planar points can be approximated within a factor of $(1 + \epsilon)$ in $O(N/(\epsilon)^{1/2})$ time.

Proof: By Theorem 32 the value chosen for K in the above algorithm is sufficient for the ϵ-approximation of the diameter. Theorem 33 establishes that $K = O(N/(\epsilon)^{1/2})$. Since the most expensive step of the algorithm is Step 2, we can compute the ϵ-approximate diameter in $O(KN) = O(N/(\epsilon)^{1/2})$ time. □

6.1.2. Second Approximate Diameter Algorithm

Although the first ϵ-approximate diameter algorithm takes only time linear in N, it is also linear in $1/\epsilon^{1/2}$. The second ϵ-approximate algorithm reduces the time from $O(N/(\epsilon)^{1/2})$ to $O(N + 1/\epsilon)$ by using a transform based on a "pie-slice" diagram (Figure 6-3) and the floor function. (Bentley, Faust, and Preparata [7] describe a different ϵ-approximate planar diameter algorithm that also runs in time $O(N + 1/\epsilon)$.) Bentley, Weide, and Yao [18] have used a simple "pie-slice" diagram for their Voronoi diagram algorithm and Weide [99] has used the floor function to speed up some of the algorithms in his thesis.

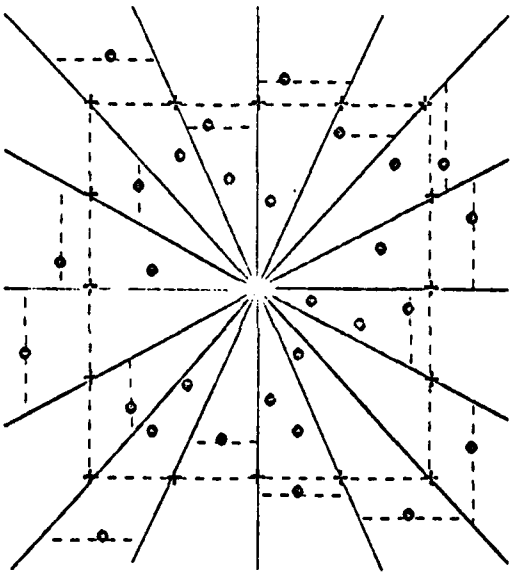


Figure 6-3: A "pie-slice" diagram.

The "pie-slice" diagram enables us to find efficiently a small subset S' of the N points of S such that the diameter of S' approximates the diameter of S. The center of the diagram, where the K "slices" meet, can be any point on or within the convex hull of S. (The slices do not each cover an angle of $2\pi/K$ radians, however, because the computation of which slice a point lies in would then require inverse

trigonometric functions. The slices are instead chosen to divide uniformly the *slopes* in each of the octants, leading to simpler computations.) From each of the slices we choose one point to insert in the subset S'. As illustrated in Figure 6-3, the point that we choose is either the farthest left, farthest right, highest, or the lowest, depending upon the orientation of the slice. We now determine how large K must be to ensure that the diameter of S' is within a factor of $(1 + \epsilon)$ of the diameter of S.

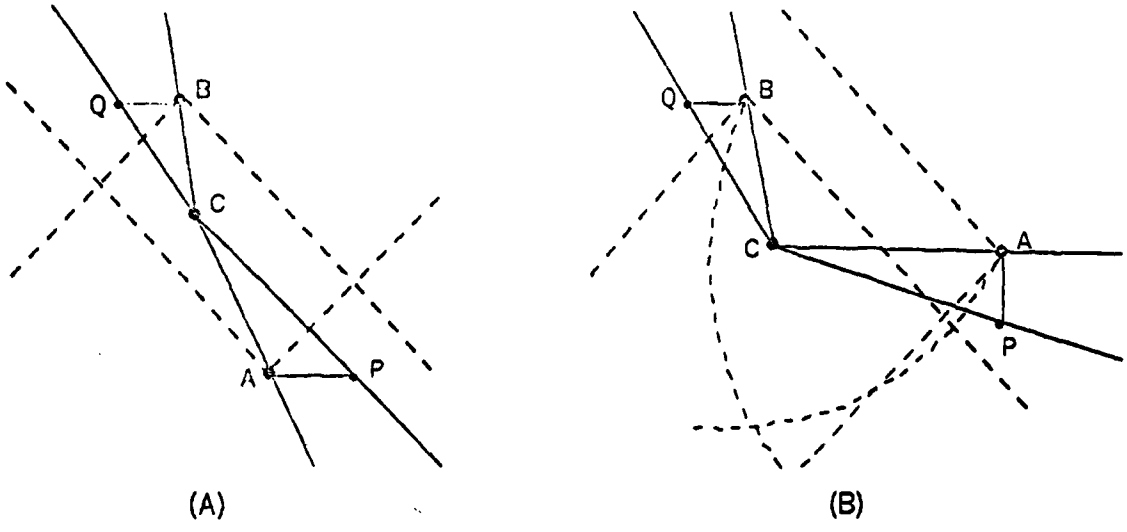


Figure 6-4: The two cases for choice of diameter with the "pie-slice" algorithm.

Theorem 35: If DIAM is the diameter of a set S of N planar points, then the diameter $DIAM_K$ of the points chosen from a pie-slice diagram of K slices satisfies

$$DIAM_K \leq DIAM \leq (1 + \epsilon)\, DIAM_K$$

if

$$K \geq \lceil 8\sqrt{1 + \sqrt{3}/2}\,(1 + \sqrt{1 + \epsilon}) / \epsilon \rceil.$$

Proof: Figure 6-4 illustrates the two possible worst cases. In both cases the Euclidean diameter is determined by points P and Q but the approximation algorithm chooses points A and B instead. The lines AP and BQ, however, are parallel for case (A) but perpendicular for case (B). The dashed lines in both figures outline the possible locations of the center C

of the pie-slice diagram.

In case 6-4 (A) we have

$$P = (P_x, P_y) = (A_x - \lambda(A_y - C_y), A_y) \text{ and}$$
$$Q = (Q_x, Q_y) = (B_x - \lambda(B_y - C_y), B_y),$$

where

$$\lambda = \delta / K.$$

Letting $DX = A_x - B_x$ and $DY = A_y - B_y$,

$$(1 + \epsilon)^2 = \frac{D(P,Q)^2}{D(A,B)^2} = \frac{(DX - \lambda DY)^2 + DY^2}{DX^2 + DY^2}.$$

Equation (25) is maximized as $DY / DX \to \infty$. The worst case for Figure 6-4 (A) is thus

$$K = \lceil \delta / \epsilon^{1/2} \rceil. \tag{26}$$

For case 6-4 (B) we have

$$P = (P_x, P_y) = (A_x, A_y - \lambda(A_x - C_x)), \text{ and}$$
$$Q = (Q_x, Q_y) = (B_x - \lambda(B_y - C_y), B_y). \tag{27}$$

Although the coordinates of point C cancelled out for case (A), case (B) is worst when the points A, B, and C form an equilateral triangle (Figure 6-5). We now have only to determine the worst orientation for this triangle. If C is the origin then A and B can be chosen to be two points on the unit circle centered at C;

$$A = (\cos(\theta), \sin(\theta)), \text{ and}$$
$$B = (\cos(\theta + \pi/3), \sin(\theta + \pi/3)). \tag{28}$$

For line AP to remain vertical while line BQ remains horizontal (as in Figure 6-4), we require that

$$\pi/4 \geq \theta \geq -\pi/12. \tag{29}$$

Combining Equations (27) and (28) we obtain

$$(1+\epsilon)^2 = \frac{D(P,Q)^2}{D(A,B)^2} = D(P,Q)^2 \qquad (30)$$
$$= F\cos^2(\theta) + G\sin^2(\theta) + H\sin(\theta)\cos(\theta),$$

where

$$F = (4 + 6\sqrt{3}\lambda + 7\lambda^2) / 4,$$
$$G = (4 + 2\sqrt{3}\lambda + \lambda^2) / 4, \text{ and}$$
$$H = (2\lambda + \sqrt{3}\lambda^2) / 2.$$

Equation (30) is maximized at

$$\tan(2\theta) = 3^{-1/2},$$

which is satisfied in the range of Equation (29) at

$$\theta = \pi/12. \qquad (31)$$

Plugging Equation (31) into Equation (30) we have

$$K = \lceil 8\sqrt{1+\sqrt{3/2}}\,(1+\sqrt{1+\epsilon}) / \epsilon \rceil \qquad (32)$$

Since the value of K for case (A) (Equation (26)) is only linear in $1/\epsilon^{1/2}$, the worst case is case (B), with K defined by Equation (32). □

With a little calculation we have

Theorem 36: If K is defined by Equation (32), then as $\epsilon \to 0$,

$$K \to \lceil 16\sqrt{1+\sqrt{3/2}} / \epsilon \rceil.$$

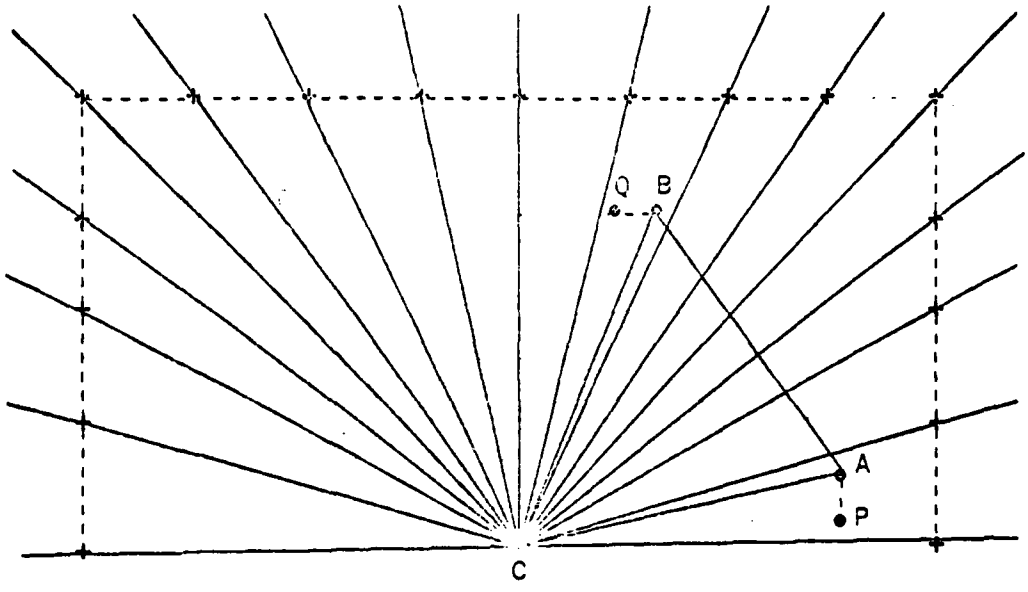


Figure 6-5: Worst case for second approximation algorithm.

Second Approximate Diameter Algorithm

1. Pick an arbitrary point from the set of N points to serve as the center of the "pie-slice" diagram. (Actually, any point within or on the convex hull of the N points can serve as the center of the diagram.) For simplicity of exposition we will assume that the center is the origin.

2. For an ϵ-approximation to the Euclidean diameter, let the number of regions in the diagram be

$$K = \lceil 4\sqrt{1 + \sqrt{3}/2}\,(1 + \sqrt{1 + 2\epsilon}) / \epsilon \rceil .$$

3. For each of the N points (x_i,y_i) (a) determine which bucket j the point falls in, and (b) compare x_i (or y_i, depending on the bucket j) with the most extreme value yet found for bucket j. We can easily do part (b) in constant time, but to avoid a binary search of O(log K) steps in part (a) we must use the floor function. For the first octant the formula is

$$\text{BUCKET} \leftarrow \lfloor (K/8)\,(y/x) \rfloor .$$

For the other octants the formula is similar.

4. Find the exact diameter $DIAM_K'$ of the K extreme points (plus the central point of the diagram) determined in Step 3. Using a Graham [48] scan to construct the convex hull and then a Shamos [91] scan to find the hull's diameter, we can compute $DIAM_K'$ in O(K) time. Return

$$DIAM_K = (1 + \epsilon)\, DIAM_K'$$

as the estimated diameter.

Theorem 37: An approximation $DIAM_K$ of the Euclidean diameter of N planar points that satisfies

$$DIAM / (1 + \epsilon) \leq DIAM_K \leq (1 + \epsilon)\, DIAM$$

can be computed in $O(N + 1/\epsilon)$ time.

Proof: Theorem 35 defines the value of K required to obtain an ϵ-approximate diameter from a "pie-slice" diagram and Theorem 36 shows that this function is $O(N + 1/\epsilon)$. □

6.2. Fitting Points on a Hemisphere

We can sometimes solve a problem involving points on a sphere similarly to the corresponding problem involving points in a plane. For example, the spherical convex hull of N spherical points is similar to a planar convex hull provided that the N spherical points all lie within a hemisphere. If, however, there is no hemispherical cap that contains all N points, then the (interior of the) spherical convex hull is the *entire* sphere. The crucial test (for the spherical convex hull problem) is to determine if there exists a hemispherical cap that contains all N points. This problem is a special case of the problem of determining the *densest hemisphere*

determined by N points on a K-dimensional sphere. Johnson and Preparata [56] have shown the densest hemisphere problem to be NP-complete when N and K are arbitrary. (For fixed K, however, there is an $O(N^{K-1} \log N)$ time algorithm.) In this section we will confine ourselves to the (simpler) problem of determining if N spherical points can all be fit into a hemispherical cap.

There are three O(N log N) time algorithms for determining if there exists a hemisphere that contains a given set of N spherical points. One solution is to intersect the N half-spaces that contain the sphere and whose boundaries are tangent to the sphere at the N points. If the resulting polytope is unbounded, then the N points can be fit into a hemispherical cap. A second solution is to construct the dual of the spherical Voronoi diagram in O(N log N) time (Section 4.2) and test if one of the faces of the diagram can contain a hemisphere. But the most interesting solution uses a transform called a *gnomonic projection* and is due to Yuval [105]. Yuval's algorithm uses gnomonic projection to transform a problem that involves great circles on a sphere to a problem that involves straight lines in the Euclidean plane. To start simply, we will first solve the two-dimensional case and then extend our result to three and four dimensions.

6.2.1. The Two-Dimensional Case

The two-dimensional problem is to determine if there exists a semicircle that contains a given set of N circular points. There are two ways in which we can solve this in O(N) time. One way is to find the largest gap between any two consecutive points of the circle. If this gap is greater than or equal to a semicircle, then all N points can fit in the other semicircle. Gonzalez [47] describes how to find the largest gap between N unsorted points on a line in O(N) time (by using the floor function). With a small modification, his algorithm will find the largest gap on a circle and thus solve the two-dimensional problem in O(N) time.

The largest gap solution does not, however, extend easily to the three-dimensional (spherical) problem, whereas the gnomonic projection solution does. Figure 6-6 illustrates how the gnomonic projection algorithm works. Let C be

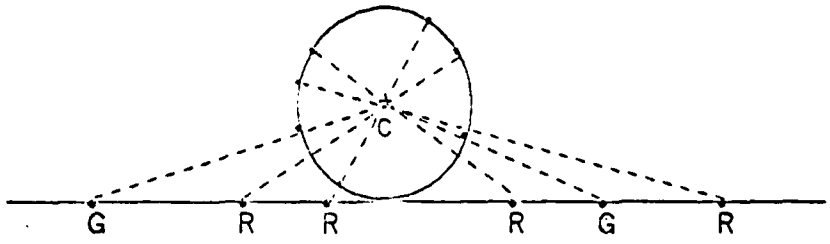


Figure G-6: Gnomonic projection of points on a circle.

the center of the circle and let L be a line tangent to the circle. The gnomonic projection transforms each circular point P to a point P′ on the tangent line L such that points P, P′, and C are collinear. This transform is not one-to-one because for each point P′ on line L, there are two points on the circle that map to P′. We can alleviate this ambiguity by giving each of the N projected points on L one of the two labels "red" or "green". If a circular point P is on the *top half* of the circle, then it maps to a point P′ labelled red. If point P is on the *bottom half* of the circle, then it maps to a point P′ labelled green. We now use gnomonic projection to transform our circular problem to a linear problem that we can solve in O(N) time.

Theorem 38: Let S be a set of N points on a circle with center C, L be a line tangent to that circle, and S′ be the set of N red and green points obtained by gnomonically projecting S onto line L. There exists a semicircle that contains all of the points of S iff the red and green points of set S′ are separable.

Proof: Part (1): Assume that there exists a semicircle T that contains the points of S. Let the endpoints of semicircle T be called T1 and T2 where T1 is on the top half of the circle and T2 is on the bottom half. Since T1 and T2 are diametrically opposite each other, they project to the same point T1′ = T2′ of line L. We will show that point T1′ = T2′ separates the red and green points of L. Any upper circular point to the left of point T1 maps to a red point that is to the right of T1′. Similarly, any upper circular point to the right of point T1 maps to a red point that is left of T1′. But for lower parts of the circle, left and right are not reversed. A lower circular point to the left of T2 maps to a green point to the left of T2 and a lower circular point to the right of T2 maps to a green point to the right of T2′. Since semicircle T contains all N points of S, there are only two cases to consider: (1) all upper points of S are to the

left of T1 and all lower points are to the left of T2, or (2) all upper points are to the right of T1 and all lower points are to the right of T2. For each of these two cases it is easy to see by the above observations that the point T1' = T2' separates the red and green points of L.

Part (2): Assume that the red and green points of L are separable. Let point T' be any point on L that separates the red and green points. There are two circular points, T1 and T2, that project to point T'. To complete our proof we must show that one of the two semicircles determined by T1 and T2 contains all of the points of S. We omit the details because they are very similar to the case analysis described in Part (1) above. □

Theorem 39: Given N points on a circle, we can determine in O(N) time if there exists a semicircle that contains all N points.

Proof: By Theorem 38 we can transform the problem of determining inclusion in a semicircle to a problem of determining separability of red and green points on a line. Since we can determine if the red and green points are separable by simply comparing the min and max of the red and green points, we can solve the two-dimensional problem in O(N) time. □

6.2.2. The Three-Dimensional Case

The gnomonic projection is a principal feature of the O(N log N) time algorithm for determining if N spherical points lie on a hemispherical cap. The definition of the gnomonic projection in three dimensions is a straightforward extension of the transform in two dimensions:

> Let P be a point on a sphere with center C and let plane L be tangent to the sphere. The gnomonic projection of spherical point P onto plane L is the point P' such that points P, P', and C are collinear.

This transform maps great circles on a sphere to straight lines on a plane, allowing us to use known algorithms for linear objects in the Euclidean plane to solve our spherical problem.

We assign the labels red and green to the projections on plane L as in the two-dimensional case. If line segment PP' contains point C, then we label P' red. Otherwise we label P' green. We now use gnomonic projection to transform our three-dimensional problem to a two-dimensional problem that we can solve in

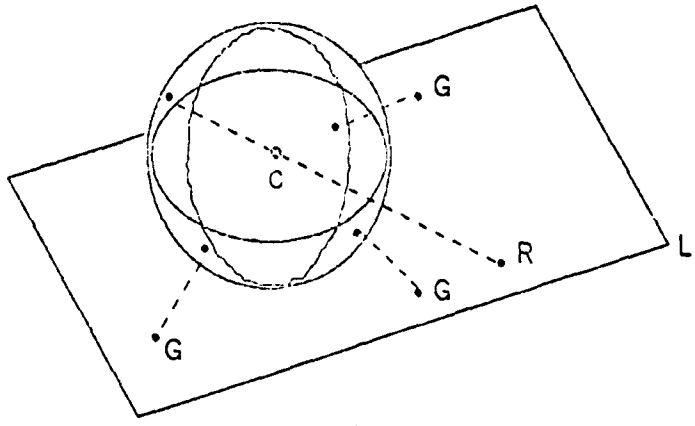


Figure G-7: Gnomonic projection of points on a sphere.

O(N log N) time.

Theorem 40: Let S be a set of N points on a sphere with center C, L be a plane that is tangent to the sphere, and S' be the set of red and green points obtained by gnomonically projecting S onto L. There exists a hemispherical cap that contains all of the points of S iff the sets of red and green points of S' are separable by a line.

Proof: The details are analogous to the proof for the two-dimensional case. Simply replace circle by sphere, semicircle by hemisphere, etc. □

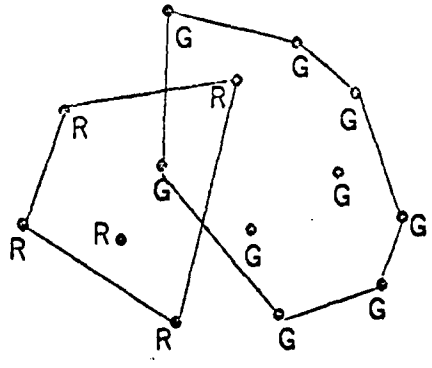


Figure G-8: Example of nonseparable sets of red and green points.

We have just reduced the problem of determining if a set of spherical points lie in a hemisphere to the problem of determining if two planar sets of points are separable by a straight line. Shamos and Hoey [94] present an O(N log N) time solution to the planar separability problem. They first make the observation that two planar sets of points are separable by a line iff the *convex hulls* of the two sets are separable by a line. The convex hulls are separable iff their intersection is empty. (See Figure 6-8 for an example of nonseparable sets of points.) Since we can construct the two convex hulls in O(N log N) time ([48]) and then intersect them in O(N) more time ([94]), we can determine separability of two planar sets of O(N) points in O(N log N) time. Since the gnomonic projection of N spherical points to N planar points costs only O(N) time, we have

> Theorem 41: Given a set S of N spherical points, we can determine in O(N log N) time whether or not all N points of S can be fit into a hemispherical cap.

6.2.3. The Four-Dimensional Case

If S is a set of N points on a four-dimensional hypersphere, then in O(N log N) time we can determine if the N points of S can be enclosed in a four-dimensional hemispherical cap. The algorithm is analogous to the algorithm for the three-dimensional case. We map the N spherical points to N red and green points in Euclidean three-space by a (four-dimensional) gnomonic projection. The N hyperspherical points fit on a hemi-hypersphere iff the red and green points are separable by a plane. We can determine separability of the red and green points in O(N log N) time. The first step is to construct the convex hulls of the red and green points in O(N log N) time ([33]), and then intersect the two convex hulls in O(N log N) time. We can construct the intersection by the algorithm of Muller and Preparata [73] or by any O(N log N) time algorithm for intersecting (three-dimensional) half-spaces. If the intersection of the two convex hulls is empty, then the N hyperspherical points of S can be enclosed by a hemi-hyperspherical cap.

6.3. Summary

This chapter presents two dissimilar sets of problems and techniques. For the approximate Euclidean diameter of a planar set of points we have been interested in a fast approximate convex hull algorithm. One approach is to successively use rotation to approximate the Euclidean metric, obtaining an $O(N/(\epsilon^{1/2}))$ time algorithm. A faster algorithm can be obtained by a partitioning approach; in this case we used wedge-shaped "pie-slices" or "buckets," yielding an $O(N + 1/\epsilon)$ time algorithm. This algorithm avoids an extra factor of $O(\log 1/\epsilon)$ by using the floor function to drop points into the appropriate "slice."

To determine if there exists a hemisphere that contains all N of a given set of spherical points we found it useful to apply a gnomonic projection. This is because a hemisphere is bounded by a great circle and the gnomonic projection maps great circles on a sphere to lines in the Euclidean plane. This enables us to use a fast planar algorithm (*separability of planar sets of points*) to solve the spherical problem in O(N log N) time. Similarly, N points on a four-dimensional sphere can be mapped to Euclidean three-space to obtain a problem (separability of two three-dimensional sets of points), which we can solve in O(N log N) time.

7. Conclusion

7.1. Transforms and Techniques

This thesis presents a new technique for construction of fast geometric algorithms -- the use of geometric transforms. Although many different kinds of transforms are presented, most fall into one of two classes

1. transforms to convert problems that are expressed in terms of circles or spheres to problems that are expressed in terms of lines or planes, and

2. transforms to convert problems that are expressed in terms of flats to problems that are expressed in terms of points.

Transforms in the first class map points to points and transforms in the second class are duality transforms. In Appendix III we list the transforms used in this thesis, their important properties, and some of their applications. In addition, Table 7-1 summarizes "typical uses" for four of the transforms that have been particularly useful.

Projection (orthographic or "radial")	Solve a K-dimensional problem as a (K-1)-dimensional problem. (Reducing dimensionality typically simplifies a problem.)
Embedding	Solve a K-dimensional problem as a (K+1)-dimensional problem. (The extra degree of freedom can sometimes allow an interpretation of the problem which is not possible in K-space.)
Point / Flat Duality	Solve a problem involving flats as a problem involving points. If half-spaces (rather than flats) are involved, then the half-spaces naturally partition into two sets (UPPER and LOWER) since each flat determines two half-spaces.
Inversion (Stereographic Projection)	Converts problems involving circles (spheres) to problems involving lines (planes). Since circles (spheres) are intimately related to the Euclidean metric, inversion may be useful in problems involving the Euclidean distance between points.

Table 7-1: Heuristics for use of some transforms.

7.2. New Results

In the preceding chapters we have presented several examples of the use of geometric transforms. Some of the major results are:

- We relate the Euclidean Voronoi diagram of a set of points to the convex hull of a transformed set of points, yielding an O(N log N) time planar algorithm, which is optimal to within a constant factor.

- We transform the problem of finding planes of support of a convex polyhedron to the problem of finding all overlapping regions among two straight-line planar graphs. This gives an O((N + K) log N) time and O(N) storage algorithm for computing the Euclidean diameter of a set of N points in three-space, where K is the number of pairs of antipodal vertices of the convex hull of the N points.

- We relate the intersection of UPPER half-spaces to the convex hull of a finite set of points. This leads to an O(N log N) time algorithm for

constructing the intersection of N UPPER three-dimensional half-spaces, which is optimal to within a constant factor.[20] We apply the floor function to obtain an $O(N + 1/\epsilon)$ time algorithm for approximating the diameter of N planar points to within a factor of $1 + \epsilon$. (This and Bentley, Faust, and Preparata's [7] $O(N + 1/\epsilon)$ time planar algorithm are the best known algorithms for approximating the Euclidean diameter.)

- We transform the union of N planar disks to an intersection of N three-dimensional half-spaces, yielding an O(N log N) time algorithm for constructing the union of N planar disks.

- There are also a number of minor but new results:

  * Location of an arbitrary point in a set of N planar disks in O(log N) time with only O(N log N) preprocessing time and O(N) storage (Section 3.2.5).

  * O(N log N) time algorithms for constructing closest and farthest point Voronoi diagrams for N spherical points and for the sides of a convex N-gon (Sections 4.2 and 4.3).

  * Determining in O(N log N) time whether N points on a four-dimensional sphere can be fit in a hemisphere (Section 6.2.3).

7.3. Open Problems

There are still several problems that remain unsolved. The list below describes several of the major outstanding problems.

- One of the major unsolved problems of this thesis is to prove an $\Omega(N \log N)$ time lower bound for computing the Euclidean diameter of N planar points under a model of computation strong enough to compute it in O(N log N) time. (Appendix II describes an approach toward such a proof.) Shamos [92] conjectures that for any metric with a continuously-turning tangent the worst-case lower bound is $\Omega(N \log N)$ time.

---

[20] This result, although similar to the work of Preparata and Muller [84], was done independently.

- What is the expected number of pairs of antipodal vertices among N three-dimensional points (under any interesting distribution of points)? (This determines the expected-time of the Euclidean diameter algorithm of Section 5.2.)

- For an approximate diameter we and Bentley, Faust, and Preparata [7] have shown an $O(N + 1/\epsilon)$ time upper bound, but the lower bound remains an open question.

- The "bucket transform" of Weide's O(N) expected-time sort [99] has already been extended to O(N) expected-time planar Voronoi diagrams by Bentley, Weide, and Yao [18]. The power of this transform derives from the use of the floor function. Most algorithms and proofs for lower bounds use models of computation that involve only comparisons between analytic functions of the input, but these results are becoming dated as the floor function is applied to more problems. Some other notable uses of the floor function are the linear-time largest gap algorithm of Gonzalez [47], linear expected-time closest-point algorithms of Yuval [104] and Rabin [86], and O(N log log N) worst-case time closest-pair algorithm of Fortune and Hopcroft [41]. These algorithms partition their problem into "buckets" and use the floor function to compute quickly for each point the bucket into which it should be dropped. What other uses can we find for the floor function in geometric algorithms?

- The algorithms for intersection of half-spaces, planar Euclidean Voronoi diagrams, spherical Voronoi diagrams, and diameter of a set of points all use the convex hull of a set of points. (Also, the algorithms that involve an intersection of half-spaces indirectly use convex hulls of sets of points.) Shamos [91] describes how a convex hull is used in isotonic regression and Silverman and Titterington [96] use a convex hull to find the smallest covering ellipse of a planar set of points. What other uses can be found for convex hulls?

7.4. Conclusion

Transforms are often useful for converting apparently difficult problems to instances of problems that are solvable by well-known methods. In this thesis we have presented a set of techniques for applying geometric transforms to geometric problems that provides another set of tools for the designer of fast geometric

algorithms. Furthermore, in the process of illustrating the application of these transforms to several problems, we have obtained several useful and interesting algorithms.

Appendix I
Finding a Good Orientation for Flats

One of the major problems with representing lines, planes, or, in general, flats in slope-intercept form is that vertical flats cannot be represented and near-vertical flats cause large round-off error. If vertical and near-vertical flats can be avoided, however, the slope-intercept form is very convenient because of the properties of the transform

$$x_K = \sum_{i=1}^{K-1} a_i x_i + a_K \rightarrow (a_1, a_2, \ldots, a_K)$$

as described in Section 3.1. In this appendix we describe how rotation of the flats can help avoid the occurrence of vertical or near-vertical flats.

There are two cases to consider:

(1) The restricted case: If the rotation is being performed to enable an intersection of N UPPER half-spaces (as in Section 3.1), then we must ensure that after rotation all N half-spaces remain UPPER half-spaces (rather than LOWER half-spaces).

(2) The general case: In general there is no restriction as in Case (1).

I.1. Case (1)

Since we want to determine what *angles* to rotate the flats through, it will be useful first to represent the flats by their angles rather than slopes. Specifically, we map the slope-intercept representation to an *angle-of-inclination* representation by

$$x_K = \sum_{i=1}^{K-1} \tan(\theta_i) x_i + a_K \rightarrow (\theta_1, \theta_2, \ldots, \theta_{K-1})$$

where $\theta_j \in [-\pi/2, \pi/2)$, $j = 1, \ldots, K-1$. In the following discussion a K-dimensional

problem of flats will therefore be treated as a K-1-dimensional problem of points. A flat will be called near-vertical if one of the K - 1 corresponding angles θ_i is within a given small angle ϵ of being vertical ($\pi/2$ or $-\pi/2$). We will also use the notation θ_{ij} for the jth coordinate (angle) for the ith flat.

We want to find a rotation $(\sigma_1, \sigma_2, \ldots, \sigma_{K-1})$ such that

$$-\pi/2 + \epsilon \leq \sigma_j + \theta_{ij} \leq \pi/2 - \epsilon, \quad \text{for } i = 1, \ldots, N, \; j = 1, \ldots, K-1. \tag{33}$$

We can easily find such a rotation, if it exists, in O(KN) time and storage. First compute the maximum and minimum angles for all K - 1 coordinates

$$\max_j = \max_i \theta_{ij} \quad \text{and} \quad \min_j = \min_i \theta_{ij}, \quad \text{for } j = 1, \ldots, K-1,$$

in O(KN) time and then determine if

$$|\pi - (\max_j - \min_j)| \geq 2\epsilon \quad \text{for } j = 1, \ldots, K-1. \tag{34}$$

If Condition (34) is satisfied then a rotation satisfying Condition (33) exists. One such rotation $(\sigma_1, \sigma_2, \ldots, \sigma_{K-1})$ is

$$\sigma_j = -(\max_j + \min_j)/2, \quad j = 1, \ldots, K-1.$$

It may not be necessary to do any rotation at all; if

$$|\pi/2 - \max_j| \geq \epsilon \quad \text{and} \quad |\pi/2 + \min_j| \geq \epsilon \quad \text{for } j = 1, \ldots, K-1 \tag{35}$$

then no rotation is required. The probability that Condition (35) is satisfied depends on the probability distribution of the points $(\theta_{i1}, \theta_{i2}, \ldots, \theta_{i,K-1})$. If the density of the distribution is uniform (and all points are independent) then Condition (35) will be satisfied with probability

$$P(\text{No rotation required}) = \left(\frac{\pi - 2\epsilon}{\pi}\right)^{N(K-1)}.$$

Condition (34) (existence of a rotation satisfying Condition (33)) is satisfied with probability

$$P(\text{rotation for Case (1) exists}) = P(\max_j - \min_j \leq \pi - 2\epsilon), \; j = 1, \ldots, K-1$$

$$= P(\max_1 - \min_1 \leq \pi - 2\epsilon)^{K-1}$$

where

$$P(\max_1-\min_1 \le \pi-2\epsilon) = \int_{x_2=0}^{\pi}\int_{x_1=\max(0,x_2-\pi+2\epsilon)}^{x_2} \frac{d^2}{dx_1 dx_2}\left(\frac{x_2-x_1}{\pi}\right)^N dx_1\, dx_2$$

$$= \left(\frac{\pi-2\epsilon}{\pi}\right)^N + \frac{2\epsilon N}{\pi}\left(\frac{\pi-2\epsilon}{\pi}\right)^{N-1}.$$

I.2. Case (2)

If we allow arbitrary rotations then the K-1-dimensional points $(\theta_1,\theta_2,\ldots,\theta_{K-1})$ may "wrap around" the boundaries of the cube of angles $[-\pi/2,\pi/2)^{K-1}$. Whereas the analysis of Case (1) required only computations related to the max's and min's for each of the K - 1 coordinates (independently), the analysis of Case (2) involves the "largest great-circle-shaped gap" between N points on a K-dimensional sphere.

The mapping that we use is (almost) the point / flat duality of Zolnowsky [106]. A flat is first translated so that it is tangent to the unit sphere (centered at the origin) and the transform is the point of tangency. Since there are two (diametrically opposite) points at which a flat of a given orientation can be tangent, there are two possible points to which a flat can map.[21] We can, of course, describe this transform in cartesian coordinates, but when we wish to speak of the *angles* of inclination it will be more useful to use the mapping

$$x_K = \sum_{i=1}^{K-1} -\cot(\theta_i)x_i + a_K \rightarrow (\theta_1,\theta_2,\ldots,\theta_{K-1}). \qquad (36)$$

where the points $(\theta_1,\theta_2,\ldots,\theta_{K-1})$ are interpreted to be normal geodesic coordinates of a K-1-dimensional sphere. (Each K-1-tuple represents two

[21] We can also express the transform as the point / flat duality of Section 3.1 followed by orthographic projection to the planes $x_k = \pm 1$, reversing the sign of all coordinates, and then gnomonic projection to the surface of the sphere.

diametrically opposite points on a sphere.) In each coordinate system this transform maps each flat to a point on the unit sphere such that the vertical flats map to points on the great circle $\theta_i = \pi/2$, $i = 1, \ldots, K-1$ (or, in cartesian coordinates, where the sphere intersects the flat $x_K = 0$). The near-vertical flats map to points near this great circle. We will present an analysis for two and three dimensions.

I.2.1. Case (2) for Lines in a Plane

Lines in the plane transform by Equation (36) to points on the top half of the circle $x^2 + y^2 = 1$ and their diametrical opposites on the bottom half. A line L is vertical iff it maps to the points $(x,y) = (1,0)$ $(\theta = 0)$ and $(x,y) = (-1,0)$ $\theta = \pi$. L is near-vertical iff it maps to points within an angle ϵ of (1,0) and (-1,0). There exists a rotation of a set of lines S such that none are near-vertical iff there exists a rotation of their corresponding points S' on the circle such that none are within an angle ϵ or (1,0) or (-1,0). This occurs iff there exists a rotation of the points (1,0) and (-1,0) such that they are not within an angle ϵ of any of the N points of S', which occurs iff there exists a gap of at least 2ϵ between adjacent points of S' on the circle. We can determine this in O(N) time by (a modification of) Gonzalez's largest gap algorithm [47]. (It should be noted, however, that Gonzalez's algorithm uses the floor function. If only analytic functions are used, then the best known algorithm for the largest gap problem takes O(N log N) time.)

I.2.2. Case (2) for Planes in Three-Space

Planes in three-space transform by Equation (36) to points on the top half of the sphere $x^2 + y^2 + z^2 = 1$ and their diametrical opposites on the bottom half. Vertical planes map to points on the great circle $\theta_1 = \theta_2 = \pi/2$ (where the sphere intersects the plane $z = 0$) and near-vertical planes lie within an angle ϵ of this great circle. There exists a rotation for a set of planes S such that no plane is near-vertical iff there exists a rotation for this great circle such that no point is within an angle ϵ of any of the points of S'. Whereas in two dimensions we obtained a problem that involved the (angular) distance between pairs of points on a circle, in

three dimensions we have a problem that involves the (angular) distance between points and a *great circle* on a sphere.

Searching for a great circle that satisfies our conditions seems more difficult than searching for a point because a great circle is a nonlocal object. We can, however, apply a point / great circle duality to obtain an equivalent point-searching problem. The transform is simple;

spherical point → great circle farthest from the point, and
great circle → the two points farthest from the great circle.

Since this duality preserves the angular distance between a point and a great circle, a spherical point P and a great circle C are an angular distance of ϵ apart iff their transforms are ϵ apart. Our new problem is thus

> Given N great circles on a sphere, find a point on the sphere that is not within an angle ϵ of any of the great circles.

This problem can be solved by the spherical analog of a planar nearest line Voronoi diagram. Unfortunately, since the N great circles partition the sphere into $\Theta(N^2)$ regions this approach will take at least $\Omega(N^2)$ time. In the worst case, it can take longer to find a good orientation for a set of N planes than it takes to intersect N three-dimensional half-spaces. If ϵ is small compared to N, however, we might be able to find a good orientation in O(N) expected-time.

Appendix II
Relation of Diameter to Empty Intersection

In this appendix we describe a relationship between the Euclidean diameter and an empty intersection problem that may lead to an $\Omega(N \log N)$ time lower bound for the diameter problem. The empty intersection problem that we are interested in is a variation of the problem of Reingold [S7]:

> Given two sets, P and Q, each of N reals in the interval [0,1), determine if there exist $p \in P$ and $q \in Q$ such that $p = q$.

We convert the empty intersection problem into a diameter problem by using inversion (Section 3.2.3). The inversion transform is determined by two parameters, the center of inversion and the radius of inversion. We shall choose the origin as the center of inversion and let the radius of inversion be one. The transform is thus

$$(x,y) \rightarrow \left(\frac{x}{x^2 + y^2}, \frac{y}{x^2 + y^2}\right).$$

For our purposes the only important property of this transform is that it transforms a line that does not pass through the origin into a circle that does pass through the origin. In particular, the line $y = 1$ maps to a circle with radius 1/2 centered at the point (0,1/2).

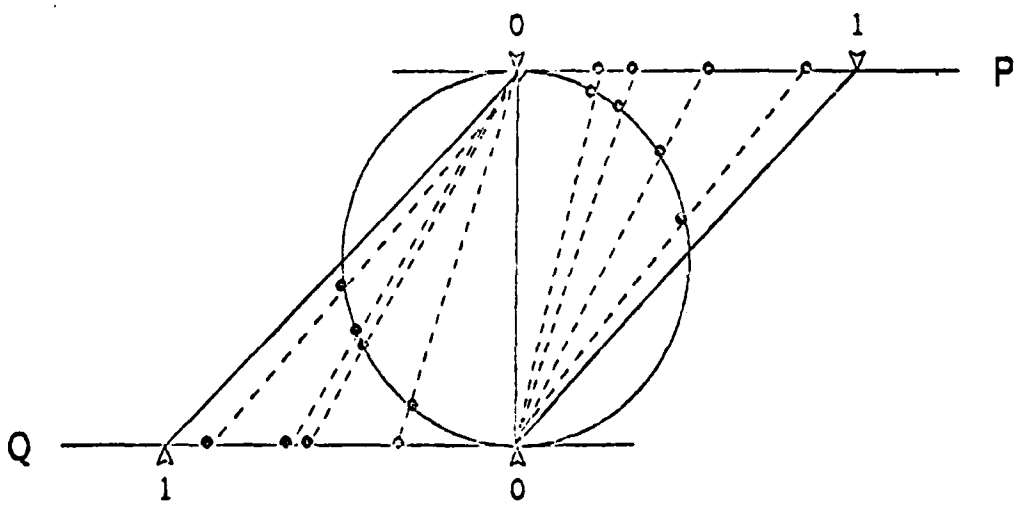


Figure 7-1: Mapping the points of sets P and Q to a circle.

Our construction first maps the N elements of set P to points on the line $y = 1$ and then applies inversion to map those points to points on the circle $x^2 + (y - 1/2)^2 = (1/2)^2$:

$$p \rightarrow (p,1) \rightarrow \left(\frac{p}{1 + p^2}, \frac{1}{1 + p^2}\right), \quad p \in P.$$

Note that since the elements of P are chosen from the set [0,1), all of the N points generated from P will lie within an arc of only one quarter of the circumference of the circle. (See Figure 7-1.) Our transform for the elements of set Q, on the other hand, generates points on a quarter-circle arc that is diametrically opposite the points from set P. More precisely, since we want to convert the empty intersection problem to a diameter problem we choose the transform for the elements of set Q so that identical elements of sets P and Q will be diametrically opposite on the circle. The transform for set Q is thus

$$q \rightarrow (-q, 0) \rightarrow \left(\frac{-q}{1 + q^2}, \frac{q^2}{1 + q^2}\right), \quad q \in Q.$$

Since all of the elements of sets P and Q are transformed to points on a circle, the maximum possible diameter of the 2N points is the diameter of the circle. This maximum is achieved iff two of the 2N points are diametrically opposite, which occurs iff there exists a $p \in P$ and $q \in Q$ such that $p = q$. We have just proved

> Theorem 42: Given a model of computation strong enough to support the inversion transform, we can transform an instance of the empty intersection problem into an N-point diameter problem in O(N) time.

To apply our lower bound for the empty intersection problem to the diameter problem, we must prove the lower bound under a model of computation strong enough to support our construction for solving the empty intersection problem as a diameter problem. Reingold's $\Omega(N \log N)$ lower bound unfortunately allows only comparisons with linear functions of the input, which is not strong enough for our purposes. One model of computation that is strong enough is a decision tree that allows only

comparisons between multivariate polynomials of degree at most K (for some fixed K ≥ 4) at internal nodes. It is important that the degree be bounded because otherwise we could solve the empty intersection problem in constant time by making the comparisons

$$\prod_{i,j}^{N} (p_i - q_j) \leq 0 \quad \text{and} \quad \prod_{i,j}^{N} (p_i - q_j) \geq 0.$$

Comparisons between bounded degree polynomial functions are sufficiently strong that we can support our construction for solving empty intersection as a diameter problem.[22] (Furthermore, we can construct O(N log N) time diameter algorithms under this model of computation.) It remains an open problem whether or not we can prove an $\Omega(N \log N)$ time lower bound for the empty intersection problem under this model of computation.

[22] It may at first appear that we cannot compute the inversion transform of our construction with finite degree polynomial functions because it involves division. That is, in a strict sense, true, but we can *simulate* it by representing a quotient p/q as an ordered pair (p,q).

Appendix III
Geometric Transforms and Applications

In this appendix we summarize the transforms used in this thesis (or used for geometric problems not described in this thesis) and list properties of and applications for each. The transforms are collected into three categories: Point-to-Point Transforms, Duality Transforms, and Miscellaneous Transforms.

III.1. Point-to-Point Transforms

Most of the point-to-point transforms fall into one of two classes; continuous and invertible transforms and projections. All continuous and invertible transforms f are potentially applicable to union or intersection problems because they satisfy

$$f(A \cup B) = f(A) \cup f(B) \quad \text{and} \quad f(A \cap B) = f(A) \cap f(B).$$

Rotation

Preserves size and shape while altering orientation.

- Convert L_∞ diameter to L_1 diameter in the Euclidean plane (Section 2.6).

General linear transform

Maps linear quantities to linear quantities while stretching and rotating.

- Generalizes problems with rectilinearly oriented lines or line segments to problems with lines or line segments at two arbitrary angles [15, 12].

- One-dimensional Johnson-Mehl crystal growth model [46] transforms to two-dimensional maxima of vectors.

- Derivation of point / flat duality from inversion and a linear transform (Section 3.3).

Orthographic projection

Reduces dimensionality by eliminating one of the cartesian coordinates and leaving the others unaffected.

- Searching a Voronoi diagram [89, 33, 31].

- Multi-dimensional divide-and-conquer algorithms that solve the merge step of a K-dimensional problem as a K-1-dimensional problem [3].

- Nearest (farthest) side diagram of a convex polygon (Section 4.3).

- Euclidean diameter of N points in three-space in O((N + K) log N) time, where K is the number of pairs of antipodal vertices on the convex hull (Section 5.2).

- Three-dimensional Chebyshev regression (Section 5.2.3).

- Least-squares isotonic regression [91].

- Least squares regression.

Perspective transformation

Lines map to lines and planes map to planes while preserving perspective information. Maps K-dimensional rays to K-1-dimensional line segments.

- Transforms a perspective projection (visibility) problem to an orthographic projection problem [97].

Radial projection (about a point)

Preserves spherical angle while reducing dimensionality.

- Dual of spherical Voronoi diagram (from convex hull) (Section 4.2).

- Spherical Voronoi diagram (from intersection of half-spaces) [22].

Gnomonic projection

A two-to-one transform that maps great circles on a sphere to lines in the

- Determine if N spherical points can fit in a hemispherical cap [105] (Section 6.2).

Stereographic projection

A conformal mapping, but it reduces the sense of the angle.

- This is a special case of inversion in three-space. (See inversion.)

Inversion

Inversion is a circular transform; circles map to circles (where a line is considered to be a circle of infinite radius). In particular,

- *A circle that passes through the center of inversion maps to a line that does not pass through the center of inversion.*

- *The interior of a circle that contains the center of inversion maps to the exterior of a circle that contains the center of inversion.*

- *Other properties are described in [36].*

- Union, intersection, subtraction of disks (Section 3.2).

- Nearest (farthest) point Voronoi diagram (Section 4.1) [23].

- Nearest (farthest) edge diagram (Section 4.3).

- Derivation of point / flat duality from inversion and linear transform (Section 3.3).

- Mapping of points on a line to points on a circle in Appendix II.

"Bucket" transform (floor function)

A discontinuous function but available as a primitive on most machines. Useful

- O(N) expected-time sort [99].

- O(N) expected-time convex hull, Voronoi diagram [18].

- O(N + 1/ε) time ε-approximation for Euclidean diameter of points (Section 6.1) [7].

Embedding into a higher dimension

Adds a degree of freedom to the problem. Lines can become planes, circles can become spheres, etc.

- Union, intersection, subtraction of disks (Section 3.2).

- Nearest (farthest) point Euclidean Voronoi diagram (Section 4.1).

- Nearest (farthest) side diagram of a convex polygon (Section 4.3).

- Lower bounds for union (intersection) of disks or half-planes (Section 3.2) or convex hull of points or triangulation of a set of points [91]. (Sort N reals by mapping them to N planar points, N disks, or N half-planes.)

- Θ(N log N) time algorithm for least-squares isotonic regression [91].

III.2. Duality Transforms

Useful for transforming problems involving nonlocal objects (such as flats) to problems involving points.

N Points in 1 Space / 1 Point in N Space Duality

Transforms an N point problem into a one point problem.

- Element-uniqueness lower bound [34].

- Epsilon-closeness lower bound [42].

- Lower bounds for sorting, insertion, finding max with analytic functions

[43].

Zolnowsky's plane / point on unit sphere duality

- O(N log N) time intersection of N half-spaces [106].

- Finding a good orientation for flats (Case (2) of Appendix I.)

Plane / Homogeneous Pluckerian coordinate duality

A more homogeneous representation of the point / flat duality below.

- O(N log N) time intersection of N half-spaces [84].

Slope Intercept form of Point / Flat duality

Preserves x_k coordinate distance between a point and a flat, thereby preserving incidence between points and flats. Preserves above/belowness between points and flats.

- O(N log N) time intersection of N UPPER (or N LOWER) half-spaces (Section 3.1).

  * Closest (farthest) side of a convex polygon (Section 4.3).

  * N points on a hemisphere: intersection of half-spaces solution (Section 6.2).

- Linear programming [28].

- Impossible three-dimensional scenes [53].

- Diameter of N points in three-space in O((N + K) log N) time (Section 5.2).

- Least-squares isotonic regression [91].

- Three-dimensional Chebyshev regression (Section 5.2.3).

- Four two-dimensional line problems: center of lines, intersection radius of lines, minimal spanning segment of lines, diameter of lines [21].

- Number of exterior points in intersection of N lines is linear [21].

Generalized (Slope-Intercept) Duality Transform

- Intersection of Half-K-Spaces (Section 3.1.5).

Circle / Point Duality

- Relation between inversion, Point / Flat duality, and convex hulls (Section 3.3).

Spherical Point (Pair) / Great Circle Duality

Preserves spherical angle between a point and a great circle.

- O(N log N) time algorithm for spherical farthest line Voronoi diagram [22].

III.3. Miscellaneous Transforms

Point to locus of a set of points transforms

- Digitization problem. Given N digitized points, find the set of all lines that have that digitization [22].

- Inclusion of N points on a hemisphere -- intersection of half-spaces solution: each point transforms to the set of points from which it is visible (in this case a half-space). (See Section 4.2.)

- Transform polyhedral obstacles to locus of forbidden positions of a reference point of the moving object [71].

Polygon to string transform

- O(N) time algorithm for similarity of polygons [72].

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, 1974).

[2] H.S. Baird, Fast Algorithms for LSI Artwork Analysis, *Journal of Design Automation & Fault-Tolerant Computing 2*, 2 (1978), 179-209.

[3] J.L. Bentley, *Divide and conquer algorithms for closest point problems in multidimensional spaces*, Ph.D. Thesis, University of North Carolina, Chapel Hill, North Carolina (1976).

[4] J.L. Bentley, Algorithms for Klee's rectangle problems, unpublished notes, Carnegie-Mellon University, 1977.

[5] J.L. Bentley, Decomposable Searching Problems, *Info. Proc. Lett. 8*, 5 (June 1979), 244-251.

[6] J.L. Bentley, Multidimensional Binary Search Trees in Database Applications, *IEEE Trans. on Software Engineering SE-5*, 4 (July 1979), 333-340.

[7] J.L. Bentley, M. Faust, and F.P. Preparata, Approximation Algorithms for Convex Hulls, in preparation.

[8] J.L. Bentley and J.H. Friedman, Fast algorithms for constructing minimal spanning trees in coordinate spaces, *IEEE Trans. Comput. C-27*, 2 (Feb. 1978), 97-105.

[9] J.L. Bentley and J.H. Friedman, A Survey of Algorithms and Data Structures for Range Searching, To appear in *ACM Comp. Surv.*.

[10] J.L. Bentley, H.T. Kung, M. Schkolnick, and C.D. Thompson, On the Average Number of Maxima in a Set of Vectors and Applications, *J. ACM 25*, 4 (Oct. 1978), 536-543.

[11] J.L. Bentley and H.A. Maurer, Efficient Worst-Case Data Structures for Range Searching, To appear in *ACTA Informatica*.

[12] J.L. Bentley and Th. Ottmann, Algorithms for Reporting and Counting Geometric Intersections, *IEEE Trans. Comp. C-28*, 9 (Sept. 1979), 643-647.

[13] J.L. Bentley and J. Saxe, Decomposable Searching Problems, *20th Annual IEEE Symposium on Foundations of Computer Science*, (1979).

[14] J.L. Bentley and M.I. Shamos, Divide and conquer in multidimensional space, pp. 220-230, *Proceedings of the Eighth Symposium on the Theory of Computing*, ACM, (May 1976).

[15] J.L. Bentley and M.I. Shamos, A Problem in Multivariate Statistics: Algorithm, Data Structure and Applications, pp. 193-201, *Proceedings of the Fifteenth Allerton Conference on Communication, Control and Computing*, (1977).

[16] J.L. Bentley and M.I. Shamos, Divide and Conquer for Linear Expected Time, *Info. Proc. Lett.* 7, 2 (Feb 1978), 87-91.

[17] J.L. Bentley, D.F. Stanat, and E.H. Williams, Jr., The complexity of near neighbor searching, *Info. Proc. Lett.* 6, 6 (Dec. 1977), 209-212.

[18] J.L. Bentley, B.W. Weide, and A.C. Yao, Optimal Expected-Time Algorithms for Closest-Point Problems, *Proceedings of the 16th Allerton Conference on Communication, Control and Computing*, (1978).

[19] J.L. Bentley and D. Wood, *An Optimal Worst-Case Algorithm for Reporting Intersections of Rectangles*, Rep. CMU-CS-79-122, (Carnegie-Mellon University Computer Science Department, May 1979).

[20] A. Borodin and I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, (American Elsevier Publishing Company, New York, 1975).

[21] K.Q. Brown, A Simple Transform for Fast Geometric Algorithms, unpublished notes.

[22] K.Q. Brown, Unpublished notes.

[23] K.Q. Brown, Voronoi Diagrams from Convex Hulls, To appear in *Info. Proc. Lett.*

[24] K.Q. Brown, *Comments on 'Algorithms for Reporting and Counting Geometric Intersections'*, Submitted for publication.

[25] D.R. Chand and S.S. Kapur, An algorithm for convex polytopes, *J. ACM 17*, 1 (Jan 1970), 78-86.

[26] D. Cheriton and R.E. Tarjan, Finding Minimum Spanning Trees, *SIAM J. Comput.* 5, 4 (Dec 1976), 724-742.

[27] D. Coppersmith, D.-T. Lee, and C.K. Wong, *Isometries Between L_1 and L_∞ Metrics in Coordinate Space*, Rep. RC 7009 (#29938), (T.J. Watson Research Center, Feb. 1978).

[28] G.B. Dantzig, *Linear Programming and Extensions*. (Princeton University Press, Princeton, New Jersey, 1963).

[29] J.C. Davis and M.J. McCullagh, *Display and Analysis of Spatial Data*, (John Wiley & Sons, 1975).

[30] L. Devroye, A Note on Finding Convex Hulls Via Maximal Vectors, to appear in *Info. Proc. Lett.*.

[31] A.K. Dewdney, *Complexity of Nearest Neighbour Searching in Three and Higher Dimensions*, Rep. 28, (Department of Computer Science, University of Western Ontario, 1977).

[32] E.W. Dijkstra, A Note on Two Problems in Connection with Graphs, *Numer. Math. 1* (1959), 269-271.

[33] D. Dobkin and R. Lipton, Multidimensional Searching Problems, *SIAM J. Comput. 5, 2* (June 1976), 181-186.

[34] D. Dobkin and R. Lipton, On the complexity of computations under varying sets of primitives, *Journal of Computer and System Sciences 18, 1* (Feb. 1979), 86-91.

[35] D. Dobkin, R. Lipton, and S.P. Reiss, *Excursions into Geometry*, Rep. 71, (Yale University Computer Science Department, 1976).

[36] C.W. Dodge, *Euclidean Geometry and Transformations*, (Addison-Wesley Publishing Co., 1972).

[37] R.L. Drysdale III and D.-T. Lee, Generalized Voronoi Diagram in the Plane, *Proceedings of the Sixteenth Allerton Conference on Communication, Control and Computing*, (1978).

[38] J.R. Dunham, D.G. Kelly, and J.W. Tolle, *Some Experimental Results Concerning the Expected Number of Pivots for Solving Randomly Generated Linear Programs*, Rep. 77 - 16, (Department of Operations Research and Systems Analysis, University of North Carolina, 1977).

[39] W. Eddy, A New Convex Hull Algorithm for Planar Sets, *ACM Transactions on*

Mathematical Software 3, 4 (Dec 1977), 398-403.

[40] R. Floyd, personal communication.

[41] S. Fortune and J. Hopcroft, A Note on Rabin's Nearest-Neighbor Algorithm, *Info. Proc. Lett. 8*, 1 (Jan. 1979), 20-23.

[42] M.L. Fredman and B. Weide, On the Complexity of Computing the Measure of $\cup[a_i,b_i]$, *Comm. ACM 21*, 7 (1978), 540-544.

[43] N. Friedman, Some Results on the Effect of Arithmetics on Comparison Problems, pp. 139-143, *13th Annual IEEE Symposium on Switching and Automata Theory (now called Foundations of Computer Science)*, IEEE, (1972).

[44] P. Gacs and L. Lovasz, *Khachian's Algorithm for Linear Programming*, Rep. STAN-CS-79-750, (Department of Computer Science, Stanford University, 1979).

[45] M.R. Garey, R.L. Graham, and D.S. Johnson, Some NP-Complete Geometric Problems, pp. 10-22, *8th Annual ACM Symposium on the Theory of Computing*, (May 1976).

[46] N. Gilbert, Random Subdivisions of Space into Crystals, *Annals of Mathematical Statistics* 33 (1962), 958-972.

[47] T. Gonzalez, *Algorithms on sets and related problems*, (Department of Computer Science, University of Oklahoma, 1975).

[48] R.L. Graham, An efficient algorithm for determining the convex hull of a planar set, *Info. Proc. Lett. 1*, 4 (June 1972), 132-133.

[49] B. Grunbaum, *Convex Polytopes*, (Wiley Interscience, 1967).

[50] F. Harary, *Graph Theory*, (Addison-Wesley Publishing Company, 1969).

[51] J.A. Hartigan, *Clustering Algorithms*, (John Wiley & Sons, 1975).

[52] J.G. Hocking and G.S. Young, *Topology*, (Addison-Wesley, 1961).

[53] D.A. Huffman, A Duality Concept for the Analysis of Polyhedral Scenes in: *Machine Intelligence 8*, E.W. Elcock and D. Michie, Ed., (Edinburgh University Press and Halsted Press (a subsidiary of John Wiley & Sons, Inc.), 1977), pp. 475-492.

[54] R.A. Jarvis, On the Identification of the Convex Hull of a Finite Set of Points in the Plane, *Info. Proc. Lett. 2* (1973), 18-21.

[55] R. Jeroslow, The Simplex Algorithm with the Pivot Rule of Maximizing Criterion Improvement, *Discrete Math. 4* (1973), 367-377.

[56] D.S. Johnson and F.P. Preparata, The Densest Hemisphere Problem, *Theoretical Computer Science* 6, 1 (1978), 93-107.

[57] T. Kanade, *A Theory of Origami World*, Rep. CMU-CS-78-144, (Carnegie-Mellon University Computer Science Department, 1978).

[58] D.G. Kelley, *On the number of iterations needed to solve a random two-dimensional linear program*, Rep. 77 - 15, (Department of Operations Research and Systems Analysis, University of North Carolina, 1977).

[59] D.G. Kirkpatrick, Efficient Computation of Continuous Skeletons, pp. 18-27, *20th Annual IEEE Symposium on Foundations of Computer Science*, (Oct. 1979).

[60] V. Klee and G. Minty, How good is the simplex algorithm? in: *Inequalities, Vol. 3*, (Academic Press, New York, 1972), pp. 159-175.

[61] L. Kleinrock, *Queueing Systems Volume 1: Theory*, (John Wiley & Sons, 1975).

[62] D.E. Knuth, Big omicron and big omega and big theta, *SIGACT News 8*, 2 (Apr-Jun 1976), 18-24.

[63] J.B. Kruskal, On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, *Proc. AMS 7* (Feb. 1956), 48-50.

[64] H.T. Kung, F. Luccio, and F.P. Preparata, On Finding the Maxima of a Set of Vectors, *J. ACM 22*, 4 (Oct. 1975), 469-476.

[65] U. Lauther, 4-Dimensional Binary Search Trees as a Means to Speed up Associative Search in Design Rule Verification of Integrated Circuits, *J. Design Automation & Fault-Tolerant Computing 2*, 3 (1978), 241-247.

[66] D.-T. Lee and F.P. Preparata, Location of a Point in a Planar Subdivision and Its Applications, *SIAM Journal of Computing 6*, 3 (Sept 1977), 594-606.

[67] D.-T. Lee and C.K. Wong, *Voronoi Diagrams in L_1 (L_∞) Metrics with*

2-Dimensional Storage Applications, Rep. RC 6848 (#29359), (IBM T.J. Watson Research Center, Nov 1977). 15 pages.

[68] D.-T. Lee and C.C. Yang, Location of Multiple Points in a Planar Subdivision, *Info. Proc. Lett.* 9, 4 (Nov. 1979), 190-193.

[69] R.J. Lipton and R.E. Tarjan, A Separator Theorem for Planar Graphs, pp. 1-10, *Waterloo Conference on Theoretical Computer Science*, (Aug 1977).

[70] R.J. Lipton and R.E. Tarjan, Applications of a Separator Theorem for Planar Graphs, *18th Annual IEEE Symposium on Foundations of Computer Science*, (1977).

[71] T. Lozano-Perez and M.A. Wesley, An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles, *Comm. ACM 22*, 10 (Oct. 1979), 560-570.

[72] G. Manacher, An application of pattern matching to a problem in geometrical complexity, *Info. Proc. Lett.* 5, 1 (1976), 6-7.

[73] D.E. Muller and F.P. Preparata, Finding the Intersection of Two Convex Polyhedra, *Theoretical Computer Science 7*, 2 (1978), 217-236.

[74] C.H. Papadimitriou and K. Steiglitz, Some Complexity Results for the Travelling Salesman Problem, pp. 1-9, *Eighth Annual ACM Symposium on the Theory of Computation*, (May 1976).

[75] D.S. Parker, *Studies in Conjugation: Huffman Tree Construction, Nonlinear Recurrences, and Permutation Networks*, Ph.D. Thesis, University of Illinois (1978). Tech. Rep. # UIUCDCS-R-78-930, UILU-ENG 78 1721.

[76] I. Pohl, A sorting problem and its complexity, *Comm. ACM 15*, 6 (1972), 462-464.

[77] F.P. Preparata, Ed., *Steps into Computational Geometry*, Rep. R-760 UILU-ENG 77-2207, (Coordinated Science Laboratory, Applied Computation Theory Group, University of Illinois, Urbana, Mar 1977).

[78] F.P. Preparata, *Steps Into Computational Geometry: Notebook II*, Rep. R-792 UILU-ENG 77-2239, (Coordinated Science Laboratory, Applied Computation Theory Group, University of Illinois, Urbana, Sept. 1977).

[79] F.P. Preparata, The Medial Axis of a Simple Polygon, pp. 443-450, *Proceedings of the Sixth Symposium on Mathematical Foundations of*

Computer Science, (Sept. 1977). Vol. 53 of *Lecture Notes in Computer Science*, G. Goos and J. Hartmanis Ed., Springer-Verlag

[80] F.P. Preparata, A New Approach to Planar Point Location, Preliminary Draft.

[81] F.P. Preparata, An Optimal Real-Time Algorithm for Planar Convex Hulls, *Comm. ACM 22*, 7 (July 1979), 402-405.

[82] F.P. Preparata, A Note on Locating a Set of Points in a Planar Subdivision, *SIAM J. Comput. 8*, 4 (Nov. 1979), 542-545.

[83] F.P. Preparata and S.J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Comm. ACM 20*, 2 (Feb 1977), 87-93.

[84] F.P. Preparata and D.E. Muller, Finding the Intersection of a Set of n Half-Spaces in Time O(n log n), *Theoretical Computer Science 8*, 1 (Feb. 1979), 45-55.

[85] R.C. Prim, Shortest Connecting Networks and Some Generalizations, *BSTJ 36*, 6 (Nov. 1957), 1389-1401.

[86] M. Rabin, Probabilistic Algorithms in: *Algorithms and Complexity*, J. Traub, Ed., (Academic Press, 1976), pp. 21-40.

[87] E.M. Reingold, On the Optimality of Some Set Algorithms, *J. ACM 19*, 4 (Oct 1972), 649-659.

[88] J.B. Saxe and J.L. Bentley, *Transforming Static Data Structures to Dynamic Structures*, Rep. CMU-CS-79-141, (Carnegie-Mellon University Computer Science Department, Sept. 1979).

[89] M.I. Shamos, Geometric Complexity, pp. 224-233, *Seventh Annual ACM Symposium on the Theory of Computation*, (1975).

[90] M.I. Shamos, Geometry and Statistics: Problems at the Interface in: *Algorithms and Complexity*, J.F. Traub, Ed., (Academic Press, 1977), pp. 251-280.

[91] M.I. Shamos, *Computational Geometry*, Ph.D. Thesis, Yale University, Department of Computer Science (May 1978). To appear as *Computational Geometry*, Springer-Verlag.

[92] M.I. Shamos, personal communication.

[93] M.I. Shamos and D. Hoey, Closest-Point Problems, pp. 151-162, *16th Annual IEEE Symposium on Foundations of Computing*, (1975).

[94] M.I. Shamos and D. Hoey, Geometric Intersection Problems, pp. 208-215, *17th Annual IEEE Symposium on Foundations of Computer Science*, (1976).

[95] M.I. Shamos and G. Yuval, Lower Bounds from Complex Function Theory, pp. 268-273, *17th IEEE Symposium on the Foundations of Computer Science*, (Oct. 1976).

[96] B.W. Silverman and D.M. Titterington, Minimum Covering Ellipses, Manuscript, University of Bath and University of Glasgow.

[97] I.E. Sutherland, R.F. Sproull, and R.A. Shumacker, A characterization of ten hidden surface algorithms, *Computing Surveys 6* (1974), 1-55.

[98] J. van Leeuwen and D. Wood, *The Measure Problem for Rectangular Ranges in d-Space*, Rep. RUU-CS-79-6, (Department of Computer Science, University of Utrecht, the Netherlands, July 1979).

[99] B.W. Weide, *Statistical Methods in Algorithm Design and Analysis*, Ph.D. Thesis, Carnegie-Mellon University, Department of Computer Science (Aug 1978).

[100] I.M. Yaglom and V.G. Boltyanskii, *Convex Figures*. (Holt, Rinehart and Winston, 1961).

[101] A.C. Yao, An O(E log log V) algorithm for finding minimum spanning trees, *Info. Proc. Lett. 4*, 1 (Sept. 1975), 21-23.

[102] A.C. Yao, *On Constructing Minimum Spanning Trees in K-Dimensional Spaces and Related Problems*, Rep. STAN-CS-77-642, (Department of Computer Science, Stanford University, Dec. 1977). 37 pages.

[103] A.C. Yao, *A Lower Bound to Finding Convex Hulls*, Rep. STAN-CS-79-733, (Department of Computer Science, Stanford University, Apr 1979).

[104] G. Yuval, Finding Nearest Neighbours, *Info. Proc. Lett. 5*, 3 (1976), 63-65.

[105] G. Yuval, personal communication.

[106] J. Zolnowsky, *Topics in Computational Geometry*, Ph.D. Thesis, Stanford University, Department of Computer Science (Jan 1978).

Index

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER: CMU-CS-80-101	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): GEOMETRIC TRANSFORMS FOR FAST GEOMETRIC ALGORITHMS	5. TYPE OF REPORT & PERIOD COVERED: Interim reptn
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): KEVIN Q. BROWN	8. CONTRACT OR GRANT NUMBER(s): N00014-76-C-0370
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Carnegie-Mellon University, Computer Science Department, Pittsburgh, PA. 15213	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS	12. REPORT DATE: Dec 79
	13. NUMBER OF PAGES: 159
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Approved for public release; distribution unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

403081